Module 2

Introduction to Programming

THEORY EXERCISE:-

1.Overview of C Programming:-

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

ANSWER : The History of C Programming

The origins of C trace back to the 1960s when computer scientists sought a programming language that could balance low-level hardware control with high-level abstraction.

1. **BCPL and B (Predecessors of C)**
   In 1966, Martin Richards developed BCPL (Basic Combined Programming Language) as a simple language for system software development. Inspired by BCPL, Ken Thompson created the B programming language in 1969 while working at Bell Labs. B was used to write early versions of the UNIX operating system but had limitations, particularly in handling data types and efficiency.
2. **The Birth of C (1972-1973)**
   Dennis Ritchie, also at Bell Labs, developed C in 1972 by improving upon B. C introduced strong data types, improved memory management, and better control structures. This made it more suitable for system programming, including writing the UNIX operating system.
3. **The Standardization of C (1978-1990s)**
   - In 1978, Brian Kernighan and Dennis Ritchie published *The C Programming Language*, which became the de facto standard for C programming.
   - In 1989, the American National Standards Institute (ANSI) established the first official standard for C, known as ANSI C (or C89).
   - Later, the International Organization for Standardization (ISO) adopted this as ISO C (C90).
   - Subsequent versions, such as C99 (1999) and C11 (2011), introduced new features, including improved memory management, inline functions, and multithreading capabilities.
4. **Modern C (2000s-Present)**
   While many programming languages have emerged, C has continued to evolve. The C17 standard (2018) brought minor improvements, and future updates aim to refine the language further. Despite its age, C remains a powerful tool in system-level programming.

Importance of C Programming:-

1. **Foundation for Modern Languages**
   C has directly influenced languages like C++, Java, Python, and JavaScript. Many programming concepts, such as syntax, memory management, and control structures, originated in C.
2. **System-Level and Embedded Programming**
   C is widely used in operating systems, embedded systems, and firmware development due to its efficiency and direct hardware interaction. The Linux kernel, Windows system components, and real-time operating systems (RTOS) are written in C.

3. **Portability and Performance**
   C is highly portable, meaning programs written in C can run on different platforms with minimal modification. Additionally, C's performance is close to assembly language, making it ideal for high-performance applications.
4. **Flexibility and Low-Level Access**
   Unlike high-level languages, C allows direct memory manipulation through pointers, making it suitable for system programming, device drivers, and embedded software.
5. **Longevity and Industry Demand**
   Despite the emergence of new programming languages, C continues to be taught in universities and used in industries such as telecommunications, automotive systems, and cybersecurity.

Why C is Still Used Today:-

1. **Embedded Systems**: Used in microcontrollers, robotics, and automotive software due to its low overhead.
2. **Operating Systems Development**: UNIX, Linux, Windows kernel components, and Android core libraries are written in C.
3. **Compilers and Interpreters**: Many modern language compilers (e.g., for Python and Java) are implemented in C.
4. **Scientific Computing and Game Development**: High-performance computing applications still favor C due to its speed and control over hardware.

2. Setting Up Environment

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

ANSWER: 1] steps to install C Compiler on windows (GCC)

1]install MinGW (minimalist GNU for windows):

Download MinGW from MinGW website.

- **Run the installer and select the gcc package under the "Basic Setup" category.**
- **Apply the changes and install.**
- **Add MinGW to your system PATH:**
  - **Open Control Panel > System > Advanced system settings > Environment Variables.**
  - **Under "System variables," find the Path variable and click "Edit."**
  - **Add the path to the MinGW bin folder (e.g., C:\MinGW\bin).**

2] Verify Installation:

- **Open Command Prompt and type gcc --version.**
- **If the GCC version is displayed, the installation is successful.**

Step 2: Set up an IDE on DevC++

1]Download and install

- Download DevC++ from SourceForge.
- Run the installer and follow the instructions.

2]Configure Compiler:

- Open DevC++ and go to Tools > Compiler Options.
- Ensure the compiler path is set to the MinGW gcc.exe (e.g., C:\MinGW\bin\gcc.exe).

3]Write and Run Code:

- Create a new C file (File > New > Source File).
- Write your code, save it with a .c extension, and compile/run it using Execute > Compile & Run.

D3] Basic Structure of a C Program:-

- **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:- A C program follows a structured format that includes essential components such as headers, the main function, comments, data types, and variables. Below is an explanation of each element, along with an example.

1. Headers (#include Directives)

- These are preprocessor directives that include necessary libraries for functions like input-output operations.
- Example: #include <stdio.h> (Standard Input-Output library).

2. Main Function (main())

- Every C program must have a main() function, as execution starts from here.
- It returns an integer value (int main() is recommended).

3. Comments

- Comments are used to improve code readability.
- **Single-line comment:** // This is a comment

Multi-line comment:

```c
/*
 This is a
 multi-line comment
*/
```

4. Data Types

- Define the type of data a variable can store.
- Common data types:
  - int (Integer)
  - float (Floating-point number)
  - char (Character)
  - double (Double precision floating-point number)

5. Variables

- Variables store data and must be declared before use.
- Example:

```c
int age = 25;
float price = 99.99;
char grade = 'A';
```

4. Operators in C

- **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 basic mathematical operations.

operator	Description	example(a = 10, b = 5)	output
+	Addition	a+b	15
-	Subtraction	a-b	5
*	Multiplication	a*b	50
/	Division	a/b	2
%	Modulus	a%b	0

2.Relational Operators

- Used to compare two values,returning true (1) or false (0)

operator	Description	example(a = 10, b = 5)	output
==	Equal to	a == b	0(false)
!=	Not equal to	a != b	1(true)
>	Greater than	a >b	1(true)
<	Less than	a <b	0(false)
>=	Greater than or equals to	a >=b	1(true)
<=	Less than or equals to	a <=b	0(false)

3.Logical Operators

- Used for logical operations, typically with boolean values (true or false).

operator	Description	example(X = 1, Y = 0)	output
&&	Logical AND	X && Y	0(false)
!	Logical NOT	!X	0(false)

4. Assignment Operators

- Used to assign values to variables.

operator	Description	Example (Equivalent To)
=	Assign	a=b
+=	Add and assign	a += b (equivalent to a = a + b)
-=	Subtract and assign	a -= b (equivalent to a = a - b)
*=	Multiply and assign	a *= b (equivalent to a = a * b)
/=	Divide and assign	a /= b (equivalent to a = a / b)
%=	Modulus and assign	a %= b (equivalent to a = a % b)

5. Increment and Decrement Operators

- It is used to increase or decrease a variable's value by 1.

operator	Description	example
++	Increment by 1	a++ (post-increment), ++a (pre-increment)
--	Decrement by 1	a–- (post-decement), –-a (pre-decrement)

6. Bitwise Operators

- Used for operations on bits

operator	symbol	Description	Example (a = 5 (0101), b = 3 (0011))
AND	&	Sets bits that are 1 in Both operands	a & b → 0101 & 0011 = 0001 (1)
XOR	^	Sets bits that are different in both operands	a ^ b → 0101 ^ 0011 = 0110 (6)
NOT	~	Inverts all bits (bitwise complement)	~a → ~0101 = 1010 (-6 in 2's complement)
Left shift	<<	Shifts bits to the left (multiplies by 2)	a << 1 → 1010 (10)
Right shift	>>	Shifts bits to the right (divides by 2)	a >> 1 → 0010 (2)

7. Conditional (Ternary) Operator

Syntax:
condition ? expression_if_true : expression_if_false;

Example:
int x = (a > b) ? a : b; // Returns the greater of a and b

5 . Control Flow Statements in C

- **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 if a condition is true.

Syntax:if (condition) { // Code to execute if condition is true

}

EXAMPLE :

```
#include <stdio.h>

int main() {
```

```
    int num = 10;
    if (num > 0) {
        printf("Number is positive.\n");
    }
    return 0;
}
```

2. if-else Statement

The if-else statement provides an alternative block of code to execute if the condition is false.

Syntax:if (condition) {

// Code to execute if condition is true

} else {

// Code to execute if condition is false

}

EXAMPLE:-

```
#include <stdio.h>
int main() {
    int num = -5;
    if (num > 0) {
        printf("Number is positive.\n");
    } else {
        printf("Number is negative or zero.\n");
    }
    return 0; }
```

3. Nested if-else Statement

A nested if-else contains another if-else inside it, allowing multiple conditions.

Syntax:`if (condition1) {`

```
if (condition2) {

// Code to execute if both conditions are true }

else { // Code to execute if condition1 is true but condition2 is false } }

else { // Code to execute if condition1 is false }
```

Example:

```
#include <stdio.h>

int main() {

    int num = 0;

    if (num > 0) {

        printf("Number is positive.\n");

    } else {

        if (num == 0) {

            printf("Number is zero.\n");

        } else {

            printf("Number is negative.\n");

        }

    }

    return 0;

}
```

4. switch Statement

The switch statement tests a variable against multiple possible values.

Syntax:

```
switch (expression) {

    case value1:

        // Code for case 1

        break;
```

```
    case value2:
        // Code for case 2
        break;
    default:
        // Code if no case matches
}
```

EXAMPLE:-

```
#include <stdio.h>
int main() {
    int choice = 2;
    switch (choice) {
        case 1:
            printf("You selected Option 1.\n");
            break;
        case 2:
            printf("You selected Option 2.\n");
            break;
        case 3:
            printf("You selected Option 3.\n");
            break;
        default:
            printf("Invalid choice.\n");
    }
    return 0;
}
```

6 . Looping in C

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

ANS.

Feature	while Loop	for Loop	do-while Loop
Initialization	Outside the loop	Inside the loop header	Outside the loop
Condition Checking	At the beginning	At the beginning	At the end
Execution Guarantee	May not execute if the condition is false initially	May not execute if the condition is false initially	Executes at least once
Use Case	When the number of iterations is unknown	When the number of iterations is known	When execution must occur at least once
Readability	Clear for indefinite loops	Best for counter-controlled loops	Useful when post-condition execution is required

When to Use Each Loop

1. **while Loop (Entry-controlled loop)**
   - Use when the number of iterations is **unknown** and depends on a condition.
   - Example: Running a program until the user provides a valid input.
   - **Best for:** Loops that rely on external conditions.

2. **for Loop (Entry-controlled loop)**

- Use when the number of iterations is **known beforehand**.
- Example: Iterating over an array or performing a task for a fixed number of times.
- **Best for:** Count-controlled loops.

3. **do-while Loop (Exit-controlled loop)**

- Use when the loop **must execute at least once**, regardless of the condition.
- Example: Showing a menu to the user at least once before checking if they want to exit.
- **Best for:** Scenarios where execution must occur at least once.

7. Loop Control Statements

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

ANS. 1. break Statement

The break statement terminates the nearest enclosing loop (for, while, or do-while) or a switch statement immediately.

Example: Using break in a Loop

```
#include <stdio.h>

int main() {

 for (int i = 1; i <= 5; i++) {

 if (i == 3) {

 break; // Exits the loop when i is 3 }

printf("%d ", i); }

 return 0; }
```

Output:

```
 1 2
```

2. continue Statement

The continue statement **skips the current iteration** and moves to the next one, without exiting the loop.

Example: Using continue in a Loop

```
#include <stdio.h>

int main() {

   for (int i = 1; i <= 5; i++) {

      if (i == 3) {

         continue;  // Skips the rest of the loop for i = 3

      }

      printf("%d ", i);

   }

   return 0;

}
```

Output:

```
1 2 4 5
```

3. goto Statement

The goto statement **jumps to a labeled statement** anywhere in the function, disrupting normal execution flow.

Example: Using goto to Jump

```
#include <stdio.h>

int main() {

 int i = 1; start: // Label

printf("%d ", i);

 i++;

if (i <= 5) {

 goto start;   // Jumps back to the label }

return 0 ;  }
```

Output:

1 2 3 4 5

8. Functions in C

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

ANS.

1. Function Declaration (Prototype)

The function is declared before main(), specifying its return type and parameters.

Syntax:

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

💡 **Example:**

```
int add(int, int);  // Function prototype
```

This tells the compiler that add() exists and takes two integers as parameters, returning an integer.

2. Function Definition

This is where the actual implementation of the function is written.

Syntax:

```
return_type function_name(parameter_list) {
   // Function body
   return value;  // (if return_type is not void)
}
```

💡 **Example:**

```
int add(int a, int b) {
   return a + b;
}
```

3. Function Call

To use the function, it must be called in main() or another function.

Syntax:

```
function_name(arguments);
```

💡 **Example:**

```
int sum = add(5, 3);  // Calling the function
```

9. Arrays in C

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

ANS. A multi-dimensional array is an array of arrays. The most common type is the two-dimensional array, which can be thought of as a table with rows and columns. Higher-dimensional arrays (e.g., 3D, 4D) are also possible.

Feature	1D ARRAY	Multi-Dimensional Array
structure	Linear list	Matrix or multi-layered structure
Indexing	Uses one index	uses two or more indices
Example	arr[5]	matrix[2][3]
Use Case	Stoing lists	Storing tabular or multi-layered data

Example

One-Dimensional

```
#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;
}
```

OUTPUT:

10 20 30 40 50

Multi-Dimensional Array:

```
#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;
}
```

OUTPUT:

1 2 3

4 5 6

10. Pointers in C

- **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. Instead of holding an actual value, it holds the address where the value is stored in memory.

A pointer is declared using the * (asterisk) symbol.

Example: Declaring and Initializing a Pointer

```
#include <stdio.h>

int main() {

   int num = 10;      // Normal integer variable

   int *ptr = &num;   // Pointer storing the address of 'num'

   printf("Value of num: %d\n", num);

   printf("Address of num: %p\n", &num);

   printf("Value of ptr (address stored in pointer): %p\n", ptr);

   printf("Value at address stored in ptr: %d\n", *ptr); // Dereferencing

   return 0;

}
```

Why Are Pointers Important in C?

1. **Efficient Memory Management**
   - Allows dynamic memory allocation (malloc, calloc, free).
2. **Pass by Reference**
   - Enables functions to modify variables outside their scope.
3. **Array and String Manipulation**
   - Efficient handling of arrays and strings.
4. **Data Structures (Linked Lists, Trees, Graphs)**
   - Essential for implementing complex data structures.
5. **Improves Performance**
   - Accessing data via pointers is often faster than using indexed access.

11. Strings in C

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

ANS. String Handling Functions in C

C provides a set of standard library functions for handling strings, which are defined in the **string.h** header file. Below are some commonly used functions:

1. strlen() – String Length

- **Purpose:** Returns the length of a string (excluding the null terminator \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: %zu\n", strlen(str));

    return 0;

}
```

- **Use Case:** Useful when you need to determine the length of a string before performing operations like copying or concatenation.

2.strcpy() - String Copy

- **Purpose:** Copies one string into another.
- **Syntax:**

```
char *strcpy(char *destination, const char *source);
```

- **Example:**

```
#include <stdio.h>

#include <string.h>

int main() {

    char src[] = "Hello, C!";

    char dest[20];  // Ensure enough space

    strcpy(dest, src);
```

```
    printf("Copied String: %s\n", dest);

    return 0;

}
```

- **Use Case:** Used for copying strings, such as when setting an initial value for a dynamically allocated string.

3. strcat() – String Concatenation

- **Purpose:** Appends one string to the end of another.
- **Syntax:**

```
char *strcat(char *destination, const char *source);
```

- **Example:**

```
#include <stdio.h>

#include <string.h>

int main() {

    char str1[30] = "Hello, ";

    char str2[] = "World!";

    strcat(str1, str2);

    printf("Concatenated String: %s\n", str1);

    return 0;

}
```

- **Use Case:** Useful when building a single string from multiple parts, such as forming file paths or messages.

4. strcmp() – String Comparison

- **Purpose:** Compares two strings lexicographically.
- **Return Value:**
  - 0 if both strings are equal.
  - A negative value if str1 is less than str2.
  - A positive value if str1 is greater than str2.
- **Syntax:**

```
int strcmp(const char *str1, const char *str2);
```

- **Example:**

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

```c
#include <string.h>

int main() {

    char str1[] = "apple";

    char str2[] = "banana";

    if (strcmp(str1, str2) == 0)

        printf("Strings are equal\n");

    else

        printf("Strings are different\n");

    return 0;

}
```

- **Use Case:** Used in sorting, searching, and checking user input in programs.

5. strchr() – Find Character in String

- **Purpose:** Searches for the first occurrence of a character in a string.
- **Syntax:**

```c
char *strchr(const char *str, int ch);
```

- **Example:**

```c
#include <stdio.h>

#include <string.h>

int main() {

    char str[] = "Hello, World!";

    char *ptr = strchr(str, 'W');

    if (ptr)

        printf("Character found at position: %ld\n", ptr - str);

    else

        printf("Character not found\n");

    return 0;

}
```

- **Use Case:** Useful for parsing input, searching for delimiters in file processing, or tokenizing strings.

12. Structures in C

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

ANS.

A **structure** in C is a user-defined data type that allows grouping multiple variables of different data types under a single name. It is useful for organizing complex data, such as records or objects.

Declaring a Structure:-

A structure is defined using the struct keyword.

```
struct StructureName {
    datatype member1;
    datatype member2;
    ...
};
```

Initializing a Structure

There are multiple ways to initialize a structure.

Method 1: Using Direct Assignment

```
struct Student s1 = {101, "Ayush", 89.5};
```

Method 2: Assigning Values Individually

```
struct Student s2;
s2.rollNo = 102;
strcpy(s2.name, "Udit"); // Use strcpy() to assign string values
s2.marks = 92.0;
```

Accessing Structure Members

Structure members are accessed using the dot (.) operator.

Example: Using a Structure

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

```c
#include <string.h>

struct Student {
    int rollNo;
    char name[50];
    float marks;
};

int main() {
    struct Student s1;
    // Assign values
    s1.rollNo = 101;
    strcpy(s1.name, "Ayush");
    s1.marks = 89.5;
    // Access and print values
    printf("Roll No: %d\n", s1.rollNo);
    printf("Name: %s\n", s1.name);
    printf("Marks: %.2f\n", s1.marks);
    return 0;
}
```

13. File Handling in C

- **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

File handling in C allows programs to store, retrieve, and manipulate data permanently on a storage device instead of keeping it in volatile memory (RAM). It enables efficient data storage and management.

Why is File Handling Important?

- **Persistent Data Storage** – Data is retained even after the program terminates.
- **Large Data Handling** – Useful for handling large amounts of data that cannot fit in memory.

- **Data Sharing** – Files allow data exchange between different programs.
- **Better Organization** – Helps structure and manage data efficiently.

1. Opening a File (fopen())

The fopen() function is used to open a file in different modes.

SYNTAX:

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

2. Closing a File (fclose())

After performing operations, the file should be closed using fclose() to free system resources.

SYNTAX:

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

3. Writing to a File

Using fprintf() – Writing Formatted Data

```
fprintf(FILE *file, "formatted text", values);
```

4. Reading from a File

Using fscanf() – Reading Formatted Data

```
fscanf(FILE *file, "format specifiers", &variables);
```

LAB EXERSICE:-

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

ANS.

1. Embedded Systems

Why C?

- C provides direct access to hardware through pointers.
- It generates efficient and fast machine code.
- It is portable across different microcontrollers and processors.

Examples of Embedded Systems Using C

- **Automobiles**: Engine control units (ECUs) in modern cars use C for real-time processing.
- **Medical Devices**: Devices like pacemakers and MRI scanners use C for performance and reliability.
- **Consumer Electronics**: TVs, washing machines, and digital cameras run firmware written in C.

2. Operating Systems (OS)

Why C?

- C provides low-level access to memory and hardware.
- It supports system programming with speed and efficiency.
- The UNIX philosophy and Linux kernel are deeply tied to C.

Examples of Operating Systems Written in C

- **Linux**: The Linux kernel is predominantly written in C.
- **Windows**: Major parts of the Windows OS are developed in C.
- **macOS and iOS**: Core components of Apple's OS are built using C.

3. Game Development

Why C?

- C provides direct memory access and efficient performance, which is critical for real-time rendering.
- It allows fine-tuned optimization required for game engines.
- Many modern game engines have C-based core components.

Examples of Game Development Using C

- **Game Engines**: The core of **Unreal Engine** and **id Tech Engine** are built using C/C++.
- **Classic Games**: Games like **Doom**, **Quake**, and **Counter-Strike** have C-based architectures.
- **Embedded Gaming Systems**: Consoles like **Nintendo Switch** and **PlayStation firmware** are developed with C.

2] Install a C compiler on your system and configure the IDE. Write your first program to print "Hello, World!" and run it

ANS.

here is the **"Hello, World!"** program

C code

```
#include <stdio.h>

int main() {

printf("Hello, World!") ;

return 0 ;  }
```

OUTPUT:

```
Hello, World!
```

3] Write a C program that includes variables, constants, and comments. Declare and use different data types (int, char, float) and display their values.

ANS.

C code

```c
#include <stdio.h>  // Standard input-output library

#define PI 3.14  // Defining a constant using #define

int main() {

   // Variable declaration

   int age = 20;        // Integer variable

   char grade = 'A';    // Character variable

   float height = 5.9;   // Floating-point variable

   const int maxMarks = 100;  // Constant using 'const' keyword

   // Displaying values using printf()

   printf("Student Details:\n");

   printf("Age: %d years\n", age);

   printf("Grade: %c\n", grade);

   printf("Height: %.1f feet\n", height);

   printf("Maximum Marks: %d\n", maxMarks);

   printf("Value of PI: %.5f\n", PI);

   return 0;

}
```

OUTPUT

```
Student Details:

Age: 20 years

Grade: A

Height: 5.9 feet

Maximum Marks: 100

Value of PI: 3.14159
```

4] Write a C program that accepts two integers from the user and performs arithmetic, relational, and logical operations on them. Display the results.

ANS.

C code

```c
#include <stdio.h>

int main() {

   int num1, num2;

   printf("Enter first integer: ");

   scanf("%d", &num1);

   printf("Enter second integer: ");

   scanf("%d", &num2);

   printf("\nArithmetic Operations:\n");

   printf("%d + %d = %d\n", num1, num2, num1 + num2);

   printf("%d - %d = %d\n", num1, num2, num1 - num2);

   printf("%d * %d = %d\n", num1, num2, num1 * num2);

   if (num2 != 0) {  // Avoid division by zero

      printf("%d / %d = %d\n", num1, num2, num1 / num2);

      printf("%d %% %d = %d\n", num1, num2, num1 % num2);

   } else {

      printf("Division and Modulo operations cannot be performed (division by zero).\n");

   }

   printf("\nRelational Operations:\n");

   printf("%d == %d : %d\n", num1, num2, num1 == num2);

   printf("%d != %d : %d\n", num1, num2, num1 != num2);

   printf("%d > %d  : %d\n", num1, num2, num1 > num2);

   printf("%d < %d  : %d\n", num1, num2, num1 < num2);

   printf("%d >= %d : %d\n", num1, num2, num1 >= num2);

   printf("%d <= %d : %d\n", num1, num2, num1 <= num2);

   printf("\nLogical Operations:\n");

   printf("(%d > 0) && (%d > 0) : %d\n", num1, num2, (num1 > 0) && (num2 > 0));
```

```
    printf("(%d > 0) || (%d > 0) : %d\n", num1, num2, (num1 > 0) || (num2 > 0));

    printf("!(%d > 0) : %d\n", num1, !(num1 > 0));

    return 0;

}
```

user enter values

```
Enter first integer: 5

Enter second integer: 3
```

OUTPUT

```
Arithmetic Operations:

5 + 3 = 8

5 - 3 = 2

5 * 3 = 15

5 / 3 = 1

5 % 3 = 2


Relational Operations:

5 == 3 : 0

5 != 3 : 1

5 > 3  : 1

5 < 3  : 0

5 >= 3 : 1

5 <= 3 : 0


Logical Operations:

(5 > 0) && (3 > 0) : 1

(5 > 0) || (3 > 0) : 1

!(5 > 0) : 0
```

5] Write a C program that accepts two integers from the user and performs arithmetic, relational, and logical operations on them. Display the results.

ANS.

CODE

```c
#include <stdio.h>  // Standard input-output library

int main() {

   int num, month;

   // Checking if a number is even or odd

   printf("Enter a number: ");

   scanf("%d", &num);

   if (num % 2 == 0) {

      printf("%d is an Even number.\n", num);

   } else {

      printf("%d is an Odd number.\n", num);

   }

   // Displaying month name based on user input

   printf("\nEnter a month number (1-12): ");

   scanf("%d", &month);

   printf("Month: ");

   switch (month) {

      case 1:  printf("January\n"); break;

      case 2:  printf("February\n"); break;

      case 3:  printf("March\n"); break;

      case 4:  printf("April\n"); break;

      case 5:  printf("May\n"); break;

      case 6:  printf("June\n"); break;

      case 7:  printf("July\n"); break;

      case 8:  printf("August\n"); break;

      case 9:  printf("September\n"); break;

      case 10: printf("October\n"); break;
```

```
        case 11: printf("November\n"); break;
        case 12: printf("December\n"); break;
        default: printf("Invalid month! Please enter a number between 1 and 12.\n");
    }
    return 0;
}
```

enter value

```
Enter a number: 15
Enter a month number (1-12): 7
```

OUTPUT

```
15 is an Odd number.
Month: July
```

6] Write a C program to print numbers from 1 to 10 using all three types of loops (while, for, do-while).

ANS.

C code

```c
#include <stdio.h>  // Standard input-output library

int main() {

   int i;  // Variable for loop iteration

   // Using 'for' loop

   printf("Using for loop:\n");

   for (i = 1; i <= 10; i++) {

      printf("%d ", i);

   }

   printf("\n\n");

   // Using 'while' loop

   printf("Using while loop:\n");

   i = 1;  // Reset i

   while (i <= 10) {

      printf("%d ", i);

      i++;

   }

   printf("\n\n");

   // Using 'do-while' loop

   printf("Using do-while loop:\n");

   i = 1;  // Reset i

   do {

      printf("%d ", i);

      i++;
```

```
    } while (i <= 10);

    printf("\n");

    return 0;

}
```

OUTPUT

```
Using for loop:

1 2 3 4 5 6 7 8 9 10

Using while loop:

1 2 3 4 5 6 7 8 9 10

Using do-while loop:

1 2 3 4 5 6 7 8 9 10
```

7] Write a C program that uses the break statement to stop printing numbers when it reaches 5. Modify the program to skip printing the number 3 using the continue statement.

ANS.

C code

```c
#include <stdio.h>  // Standard input-output library

int main() {

   int i;

   // Using 'break' to stop at 5

   printf("Using break (Stops at 5):\n");

   for (i = 1; i <= 10; i++) {

      if (i == 5) {

         break;  // Stops loop when i is 5

      }

      printf("%d ", i);

   }

   printf("\n\n");

   // Using 'continue' to skip 3

   printf("Using continue (Skips 3):\n");

   for (i = 1; i <= 10; i++) {

      if (i == 3) {

         continue;  // Skips 3, goes to next iteration

      }

      printf("%d ", i);

   }

   printf("\n");

   return 0;

}
```

OUTPUT

```
Using break (Stops at 5):

1 2 3 4

Using continue (Skips 3):

1 2 4 5 6 7 8 9 10
```

8] Write a C program that calculates the factorial of a number using a function. Include function declaration, definition, and call.

ANS.

C code

```c
#include <stdio.h>  // Standard input-output library

int main() {

   int i;

   // Using 'break' to stop at 5

   printf("Using break (Stops at 5):\n");

   for (i = 1; i <= 10; i++) {

      if (i == 5) {

         break;  // Stops loop when i is 5

      }

      printf("%d ", i);

   }

   printf("\n\n");

   // Using 'continue' to skip 3

   printf("Using continue (Skips 3):\n");

   for (i = 1; i <= 10; i++) {

      if (i == 3) {

         continue;  // Skips 3, goes to next iteration

      }

      printf("%d ", i);

   }

   printf("\n");

   return 0;

}
```

input by user

```
Enter a number to calculate its factorial: 5
```

output

```
Factorial of 5 is 120
```

9] Write a C program that stores 5 integers in a one-dimensional array and prints them. Extend this to handle a two-dimensional array (3x3 matrix) and calculate the sum of all elements.

ANS.

C code

```c
#include <stdio.h>

int main() {

  // Part 1: One-Dimensional Array

  int arr[5];  // 1D array of size 5

  printf("Enter 5 integers:\n");

  for (int i = 0; i < 5; i++) {

    scanf("%d", &arr[i]);  // Taking input

  }

  printf("The entered numbers are: ");

  for (int i = 0; i < 5; i++) {

    printf("%d ", arr[i]);  // Printing array elements

  }

  printf("\n\n");

  // Part 2: Two-Dimensional Array (3x3 matrix)

  int matrix[3][3], sum = 0;

  printf("Enter 9 elements for the 3x3 matrix:\n");

  for (int i = 0; i < 3; i++) {  // Row loop

    for (int j = 0; j < 3; j++) {  // Column loop

      scanf("%d", &matrix[i][j]);  // Taking input

      sum += matrix[i][j];  // Summing elements

    }

  }

  printf("The entered 3x3 matrix is:\n");
```

```
    for (int i = 0; i < 3; i++) {  // Printing matrix
        for (int j = 0; j < 3; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
    printf("\nSum of all elements in the matrix = %d\n", sum);  // Printing sum
    return 0;
}
```

input

```
Enter 5 integers:
1 2 3 4 5
Enter 9 elements for the 3x3 matrix:
1 2 3
4 5 6
7 8 9
```

output

```
The entered numbers are: 1 2 3 4 5

The entered 3x3 matrix is:

1 2 3

4 5 6

7 8 9

Sum of all elements in the matrix = 45
```

10] Write a C program to demonstrate pointer usage. Use a pointer to modify the value of a variable and print the result.

ANS.

c code

```c
#include <stdio.h>  // Standard input-output library

int main() {

   int num = 10;  // Declare an integer variable

   int *ptr;     // Declare a pointer variable

   ptr = &num;    // Assign the address of 'num' to the pointer

   // Print initial values

   printf("Before modification:\n");

   printf("Value of num = %d\n", num);

   printf("Address of num = %p\n", &num);

   printf("Pointer ptr stores address = %p\n", ptr);

   printf("Value pointed by ptr = %d\n\n", *ptr);

   // Modify the value of 'num' using the pointer

   *ptr = 20;

   // Print modified values

   printf("After modification using pointer:\n");

   printf("Value of num = %d\n", num);

   printf("Value pointed by ptr = %d\n", *ptr);

   return 0;

}
```

OUTPUT

```
Before modification:

Value of num = 10

Address of num = 0x7ffeefbff56c

Pointer ptr stores address = 0x7ffeefbff56c

Value pointed by ptr = 10

After modification using pointer:

Value of num = 20

Value pointed by ptr = 20
```

11]Write a C program that takes two strings from the user and concatenates them using strcat(). Display the concatenated string and its length using strlen().

ANS.

c code

```c
#include <stdio.h>

#include <string.h>

int main() {

   char str1[100], str2[100];  // Arrays to store input strings

   // User input

   printf("Enter first string: ");

   fgets(str1, sizeof(str1), stdin);  // Reads first string (including spaces)

   str1[strcspn(str1, "\n")] = 0;     // Remove trailing newline

   printf("Enter second string: ");

   fgets(str2, sizeof(str2), stdin);  // Reads second string

   str2[strcspn(str2, "\n")] = 0;     // Remove trailing newline

   // Concatenating str2 to str1

   strcat(str1, str2);

   // Display results

   printf("\nConcatenated String: %s\n", str1);

   printf("Length of Concatenated String: %lu\n", strlen(str1));

   return 0;

}
```

input

```
Enter first string: Hello,

Enter second string: World!
```

output

```
Concatenated String: Hello, World!

Length of Concatenated String: 13
```

12] Write a C program that defines a structure to store a student's details (name, roll number, and marks). Use an array of structures to store details of 3 students and print them.

ANS.

c code

```c
#include <stdio.h>

struct Student {
    char name[50];
    int rollNumber;
    float marks;
};

int main() {
    struct Student students[3];  // Array of 3 student structures

    // Taking input for 3 students
    printf("Enter details for 3 students:\n");
    for (int i = 0; i < 3; i++) {
        printf("\nEnter details for Student %d:\n", i + 1);
        printf("Name: ");
        scanf(" %[^\n]", students[i].name);  // Read full name including spaces
        printf("Roll Number: ");
        scanf("%d", &students[i].rollNumber);
        printf("Marks: ");
        scanf("%f", &students[i].marks);
    }

    // Displaying the student details
    printf("\nStudent Details:\n");
    for (int i = 0; i < 3; i++) {
        printf("\nStudent %d\n", i + 1);
```

```
        printf("Name: %s\n", students[i].name);
        printf("Roll Number: %d\n", students[i].rollNumber);
        printf("Marks: %.2f\n", students[i].marks);
    }
    return 0;
}
```

input

```
Enter details for 3 students:
Enter details for Student 1:
Name: Ayush
Roll Number: 145
Marks: 89.5
Enter details for Student 2:
Name: dhiru
Roll Number: 102
Marks: 76.2
Enter details for Student 3:
Name: udit
Roll Number: 103
Marks: 92.8
```

output

```
Student Details:

Student 1

Name: Ayush

Roll Number: 145

Marks: 89.50

Student 2

Name: dhiru

Roll Number: 102

Marks: 76.20

Student 3

Name: udit

Roll Number: 103

Marks: 92.80
```

13] Write a C program to create a file, write a string into it, close the file, then open the file again to read and display its contents.

ANS.

c code

```c
#include <stdio.h>  // Standard input-output library

int main() {

    FILE *file;  // File pointer

    char text[] = "Hello, I'm AYUSH";

    char buffer[100];  // Buffer to store read content

    // Step 1: Create and Write to the File

    file = fopen("example.txt", "w");  // Open file in write mode

    if (file == NULL) {

        printf("Error opening file!\n");

        return 1;

    }

    fprintf(file, "%s", text);  // Write string to file

    fclose(file);  // Close the file after writing

    printf("Data written to file successfully.\n");

    // Step 2: Open and Read the File

    file = fopen("example.txt", "r");  // Open file in read mode

    if (file == NULL) {

        printf("Error opening file!\n");

        return 1;

    }

    fgets(buffer, sizeof(buffer), file);  // Read content from file

    printf("File content: %s\n", buffer);

    fclose(file);  // Close the file after reading
```

```
    return 0;

}
```

output

```
Data written to file successfully.

File content: Hello, I'm AYUSH
```