# Module 2 – Introduction to Programming

## Overview of C Programming

**Q1. Write an essay covering the history and evolution of C programming. Explain its importance and why it is still used today.**

The history of C dates back to the 1960s and 1970s, rooted in the development of the UNIX operating system. Before C, programmers used assembly language or earlier high-level languages like B and BCPL.

1. **BCPL and B Language**
   In 1967, Martin Richards developed BCPL (Basic Combined Programming Language) for writing system software. Inspired by BCPL, Ken Thompson at Bell Labs created the B language in 1969 to develop the early versions of UNIX on the DEC PDP-7 computer.
2. **Creation of C (1972)**
   Dennis Ritchie, also at Bell Labs, developed C in 1972 by improving upon the B language. C introduced data types, structures, and other features, enabling programmers to write more robust and complex software. It was used to rewrite the UNIX operating system, making it one of the first operating systems written in a high-level language.
3. **Standardization (1980s)**
   C grew in popularity, leading to the need for standardization. The American National Standards Institute (ANSI) formed a committee in 1983 to standardize C, resulting in ANSI C (also known as C89 or C90), which became the widely accepted standard.
4. **Later Developments**
   Over time, additional standards emerged, such as C99 (1999) and C11 (2011), which added features like inline functions, new data types, and improved multi-threading support. These updates kept C relevant in the changing landscape of programming.

**Importance of C Programming**

1. **Performance and Efficiency**
   C provides low-level access to memory and minimal runtime overhead, making it ideal for systems where performance and efficiency are critical, such as operating systems and embedded devices.
2. **Portability**
   C programs can be compiled and run on different types of machines with minimal changes, making it a portable language and contributing to its wide adoption across platforms.

**Operating Systems**: Most parts of UNIX, Linux, and Windows are written in C.

**Embedded Systems**: Devices like microcontrollers, routers, and IoT devices rely on C due to its closeness to hardware.

**Game Development**: Game engines and performance-critical parts are often written in C or c++.

**Compilers and Interpreters**: Many are developed using C due to its speed and reliability.

**Q2. Research and provide three real-world applications where C programming is extensively used, such as in embedded systems, operating systems, or game development.**

1. Embedded Systems

Application Example: Automotive Control Systems

- Where it's used: Engine Control Units (ECUs), Anti-lock Braking Systems (ABS), Airbag systems, and infotainment systems in cars.
- Why C is used:
  - Offers direct hardware access through pointers and memory management.
  - Produces efficient and fast code, essential for real-time performance.
  - Most microcontrollers and processors have C-based compilers.

- Example: The software in a Bosch ECU, a critical part of modern cars, is programmed using C to handle fuel injection, ignition timing, and emissions control.

2. Operating Systems

Application Example: Linux Operating System

- Where it’s used: Linux kernel, UNIX systems, Windows system components.
- Why C is used:
    - Offers low-level access to system memory and hardware.
    - Enables creation of high-performance system software.
    - Portable code, easily compiled on various hardware platforms.
- Example: The Linux kernel, which powers everything from Android phones to supercomputers, is written mostly in C.

3. Game Development

Application Example: Game Engines (e.g., Unreal Engine)

- Where it’s used: Core engines, graphics rendering systems, physics engines.
- Why C is used:
    - Delivers high-speed execution, crucial for real-time rendering and gaming performance.
    - Provides fine control over hardware and system resources.
    - Often paired with C++ for object-oriented features and performance optimization.
- Example: The core of the Unreal Engine, one of the world’s leading game engines used in titles like *Fortnite* and *Gears of War*, is written in a combination of C and C++.

**2.Setting Up Environment**

**Q1. 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.**

Step 1: Install GCC (C Compiler)

GCC (GNU Compiler Collection) is a widely used C compiler.

Option A: Install TDM-GCC or MinGW (Windows)

1. Go to https://jmeubank.github.io/tdm-gcc/
2. Download the installer (e.g., tdm-gcc.exe).
3. Run the installer and follow the instructions.
4. Make sure to select "C/C++ Compiler" during installation.
5. After installation, verify GCC installation:
    - Open Command Prompt
    - Type: gcc --version
    - You should see GCC version information.

Step 2: Choose and Install an IDE

Option A: Dev-C++ (Beginner Friendly)

1. Download from: https://sourceforge.net/projects/orwelldevcpp/
2. Run the installer and follow the setup instructions.
3. Dev-C++ includes GCC by default (you don’t need to install it separately).
4. Create a new C project:
    - Go to File > New > Project
    - Choose Console Application > C
    - Start coding!

Option B: VS Code (Advanced & Modern)

1. Download from: https://code.visualstudio.com/
2. Install GCC separately (use TDM-GCC or MinGW as above).

3. Open VS Code, then:
    - Install C/C++ Extension:
        - Go to Extensions (Ctrl+Shift+X)
        - Search for "C/C++" by Microsoft and install.
    - Install Code Runner (optional for quick execution).
4. Set up build tasks:
    - Press Ctrl+Shift+B
    - VS Code will offer to configure tasks.json for GCC.
5. Write a .c file and compile it using Terminal > Run Build Task.

Option C: Code::Blocks

1. Download from: http://www.codeblocks.org/downloads/26
2. Choose "codeblocks-XXmingw-setup.exe" – includes the GCC compiler.
3. Run the installer.
4. Launch Code::Blocks, and it will auto-detect the compiler.
5. Create a new project:
    - Go to File > New > Project > Console Application > C
    - Follow the wizard and start coding.

**Q2. Install a C compiler on your system and configure the IDE. Write your first program to print "Hello, World!" and run it.**

Step 1: install a c compiler

- Download from : [http://jmeubank.github.io./tdm-c/](http://jmeubank.github.io./tdm-c/)
- Run the installer
- Choose “tdm-gcc 64-bit” in the setup wizard
- Finish installation
- Test installation
  Open command prompt
  Type gcc—version

Step 2: install an IDE

- Download from: https//sourceforge.net/projects/orwelldevcpp/
- Install it with default settings
- Open dev-c++ and create a new project
  File> new > project console application> c
  Name your project and save it

**Program**

**#include<stdio.h>**

**Main(){**

**Printf("hello world");**

**}**

**3. Basic Structure of a C Program**

**Q1. Explain the basic structure of a C program, including headers, main function, comments, data types, and variables. Provide examples.**
explanation of components

1. Header file
   - Start with #include
   - Used to include libraries that contain built-in functions

Example:

```
    #include<stdio.h> //standard input/output function like  printf and scanf
```

2. Comments
   - Single line comments
   - Multi line comments

Example:

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

 /*

   This is a
```

multi-line comment

*/

3. Main function
    - Every C program must have a main() function.
    - Execution starts from here.

Example:

```
        int main() {
   // Code here
   return 0;
}
```

4. Data types

| Data Type | Description | Example |
|---|---|---|
| int | Integer numbers | int age = 20; |
| float | Decimal numbers | float price = 99.5; |
| char | Single characters | char grade = 'A'; |
| double | Double precision float | double pi = 3.14159; |

5. Variables
    - Variables are named memory locations.
    - Must be declared with a data type.

Example:

```
        int marks = 95;
float weight = 62.5;
char initial = 'S';
```

**Q2.Write a C program that includes variables, constants, and comments. Declare and use different data types (int, char, float) and display their values.**

```
#include <stdio.h>
int main() {
   const float PI = 3.14159;
   int age = 25;
   char grade = 'A';
   float height = 5.9;
   printf("Student Details:\n");
   printf("Age: %d\n", age);
   printf("Grade: %c\n", grade);
   printf("Height: %.1f feet\n", height);
   printf("Value of PI: %.5f\n", PI);

   return 0;
}
```

**4.Operators in C**

**Q1. Write notes explaining each type of operator in C: arithmetic, relational, logical, assignment, increment/decrement, bitwise, and conditional operators.**

1. **Arithmetic Operators**

| **Operator** | **Meaning** | **Example** | **Result** |
|---|---|---|---|
| + | Addition | a + b | Adds a and b |
| - | Subtraction | a - b | Subtracts b from a |
| * | Multiplication | a * b | Multiplies a and b |

| Operator | Meaning | Example | Result |
|---|---|---|---|
| / | Division | a / b | Divides a by b |
| % | Modulus (Remainder) | a % b | Remainder of a/b |

2. **Relational Operators**

| Operator | Meaning | Example |
|---|---|---|
| == | Equal to | a == b |
| != | Not equal to | a != b |
| > | Greater than | a > b |
| < | Less than | a < b |
| >= | Greater than or equal to | a >= b |
| <= | Less than or equal to | a <= b |

3. **Logical Operators**

| Operator | Meaning | Example | Description |
|---|---|---|---|
| && | Logical AND | a > 0 && b < 10 | True if both conditions are true |
| ` | | ` | Logical OR |
| ! | Logical NOT | !a | True if a is false |

4. **Assignment Operators**

| Operator | Meaning | Example | Equivalent To |
|---|---|---|---|
| = | Assign | a = 10 | - |

| Operator | Meaning | Example | Equivalent To |
|---|---|---|---|
| += | Add and assign | a += 5 | a = a + 5 |
| -= | Subtract and assign | a -= 3 | a = a - 3 |
| *= | Multiply and assign | a *= 2 | a = a * 2 |
| /= | Divide and assign | a /= 4 | a = a / 4 |
| %= | Modulus and assign | a %= 3 | a = a % 3 |

## 5. Increment and Decrement Operators

| Operator | Meaning | Example | Description |
|---|---|---|---|
| ++ | Increment by 1 | a++ / ++a | Increases a by 1 |
| -- | Decrement by 1 | a-- / --a | Decreases a by 1 |

## 6. Bitwise Operators

| Operator | Meaning | Example | Description |
|---|---|---|---|
| & | AND | a & b | Bitwise AND |
| ` | ` | OR | `a |
| ^ | XOR | a ^ b | Bitwise exclusive OR |
| ~ | NOT | ~a | Bitwise complement |
| << | Left Shift | a << 1 | Shifts bits to the left |
| >> | Right Shift | a >> 1 | Shifts bits to the right |

## 7. Conditional (Ternary) Operator

| Syntax | Meaning |
|---|---|
| condition ? true : false | Returns value based on condition |

**Q2. Write a C program that accepts two integers from the user and performs arithmetic, relational, and logical operations on them. Display the results.**

```c
#include <stdio.h>

int main() {

   int a, b;

   printf("Enter first integer (a): ");

   scanf("%d", &a);

   printf("Enter second integer (b): ");

   scanf("%d", &b);

   printf("\n--- Arithmetic Operations ---\n");

   printf("a + b = %d\n", a + b);

   printf("a - b = %d\n", a - b);

   printf("a * b = %d\n", a * b);

   if (b != 0) {

      printf("a / b = %d\n", a / b);

      printf("a %% b = %d\n", a % b);

   } else {

      printf("Division and modulus by zero are not allowed.\n");

   }

   printf("\n--- Relational Operations ---\n");

   printf("a == b: %d\n", a == b);

   printf("a != b: %d\n", a != b);

   printf("a > b : %d\n", a > b);

   printf("a < b : %d\n", a < b);

   printf("a >= b: %d\n", a >= b);
```

```
    printf("a <= b: %d\n", a <= b);

    printf("\n--- Logical Operations ---\n");

    printf("a && b = %d\n", a && b);

    printf("a || b = %d\n", a || b);

    printf("!a = %d\n", !a);

    printf("!b = %d\n", !b);


    return 0;

}
```

## 5.Control Flow Statements in C

**Q1.Explain decision-making statements in C (if, else, nested if-else, switch). Provide examples of each.**

### 1. if Statement

Executes a block of code **only if** a condition is true.

Example:-

```
int num = 10;

if (num > 0) {

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

}
```

### 2. if...else Statement

Executes one block if the condition is true, another if false.

Example:-

```
int num = -5;

if (num >= 0) {
    printf("Number is non-negative.\n");
} else {
    printf("Number is negative.\n");
}
```

**3. Nested if...else Statement**

An if or else block contains another if...else.

Example:-

```
int num = 25;

if (num > 0) {
    if (num % 2 == 0) {
        printf("Positive even number.\n");
    } else {
        printf("Positive odd number.\n");
    }
} else {
    printf("Number is not positive.\n");
}
```

**4. switch Statement**

Allows multi-way branching based on the value of a variable/expression.

Example:-

```
int choice = 2;

switch (choice) {
    case 1:
        printf("Option 1 selected.\n");
        break;
    case 2:
        printf("Option 2 selected.\n");
        break;
    default:
        printf("Invalid option.\n");
}
```

**Q2.Write a C program to check if a number is even or odd using an if-else statement. Extend the program using a switch statement to display the month name based on the user's input (1 for January, 2 for February, etc.).**

```
#include <stdio.h>
int main() {
    int number, month;
    printf("Enter an integer: ");
    scanf("%d", &number);
    if (number % 2 == 0) {
        printf("%d is even.\n", number);
    } else {
        printf("%d is odd.\n", number);
    }
    printf("\nEnter a number (1 to 12) to get the month name: ");
    scanf("%d", &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 number! Please enter 1 to 12.\n");
    }
    return 0;
}
```

**6.Looping in C**

**Q1.Compare and contrast while loops, for loops, and do-while loops. Explain the scenarios in which each loop is most appropriate.**

1. while Loop

Key Features:

- Entry-controlled loop (condition is checked before execution).
- Loop may not execute at all if the condition is false initially.

Example:-

```
int i = 1;
while (i <= 5) {
   printf("%d ", i);
   i++;
}
```

2. for Loop

Key Features:

- Entry-controlled loop.
- All loop control statements are in one line (compact and readable).
- Suitable when the number of iterations is known.

Example:-

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

3. do-while Loop

Key Features:

- Exit-controlled loop (condition is checked after the loop body).
- Always executes at least once, regardless of the condition.

Example:-

```
int i = 1;
do {
   printf("%d ", i);
   i++;
} while (i <= 5);
```

**Q2.Write a C program to print numbers from 1 to 10 using all three types of loops (while, for, do-while).**

```
#include <stdio.h>
int main() {
   int i;
   printf("Using while loop:\n");
   i = 1;
   while (i <= 10) {
      printf("%d ", i);
      i++;
   }
   printf("\n");
   printf("Using for loop:\n");
   for (i = 1; i <= 10; i++) {
      printf("%d ", i);
   }
   printf("\n");
   printf("Using do-while loop:\n");
   i = 1;
   do {
      printf("%d ", i);
      i++;
   } while (i <= 10);
   printf("\n");
   return 0;
}
```

## 7.Loop Control Statements

**Q1. Explain the use of break, continue, and goto statements in C. Provide examples of each.**

**1. break Statement**

**Use:**

- Immediately exits a loop (for, while, do-while) or a switch statement.
- Control moves to the first statement after the loop or switch block.

Example:-

```
#include <stdio.h>
int main() {
   for (int i = 1; i <= 10; i++) {
      if (i == 5)
         break;
      printf("%d ", i);
   }
   return 0;
}
```

**2. continue Statement**

**Use:**

- Skips the current iteration of the loop and jumps to the next iteration.
- Useful when you want to skip some values but continue looping.

Example:-

```
#include <stdio.h>
int main() {
   for (int i = 1; i <= 5; i++) {
       if (i == 3)
           continue;  // Skip printing when i is 3
       printf("%d ", i);
   }
   return 0;
}
```

**3. goto Statement**

**Use:**

- Jumps unconditionally to a labeled statement within the same function.
- Generally discouraged due to its ability to create complex and hard-to-follow code (spaghetti code).
- Can be useful in breaking out of deeply nested loops.

Example:-

```
#include <stdio.h>
int main() {
    int i = 1;
    start:
    if (i <= 5) {
        printf("%d ", i);
        i++;
        goto start;  // Jumps back to the label
    }
    return 0;
}
```

**Q2. 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.**

Use break to stop printing numbers when it reaches 5

```
#include <stdio.h>

int main() {

   int i;

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

      if (i == 5) {

         break;

      }

      printf("%d ", i);

   }

   return 0;

}
```

Modify to skip printing number 3 using continue

```
#include <stdio.h>

int main() {

   int i;

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

      if (i == 3) {

         continue;  // Skip printing when i is 3

      }

      if (i == 5) {

         break;     // Stop the loop when i is 5

      }
```

```
        printf("%d ", i);
    }
    return 0;
}
```

**8.Functions in C**

**Q1. What are functions in C? Explain function declaration, definition, and how to call a function. Provide examples.**

A **function** in C is a **block of reusable code** that performs a specific task. Functions help divide a program into smaller, manageable parts, improve code reusability, and make debugging easier.

**Function Declaration (Prototype)**

- Tells the compiler about the function name, return type, and parameters before it is used.

**Function Definition**

- Contains the actual code of the function.

**Function Call**

- Executes the function and passes control along with any required values.

Example: Program Using a Function to Add Two Numbers

```
#include <stdio.h>

int add(int a, int b);

int main() {
    int x = 10, y = 20, sum;
    sum = add(x, y);
    printf("Sum = %d\n", sum);
    return 0;
}
int add(int a, int b) {
    return a + b;
}
```

**Q2. Write a C program that calculates the factorial of a number using a function. Include function declaration, definition, and call.**

```
#include <stdio.h>
long long factorial(int n);
int main() {
    int num;
    long long result;
    printf("Enter a number: ");
    scanf("%d", &num);
    if (num < 0) {
        printf("Factorial is not defined for negative numbers.\n");
    } else {
        result = factorial(num);
        printf("Factorial of %d = %lld\n", num, result);
    }
    return 0;
}
long long factorial(int n) {
    long long fact = 1;
    for (int i = 1; i <= n; i++) {
        fact *= i; // fact = fact * i
    }
    return fact;
}
```

**Function Declaration**

long long factorial(int n);

**Function Definition**

long long factorial(int n) { ... }

**Function Call**

result = factorial(num);

**9.Arrays in C**

**Q1. Explain the concept of arrays in C. Differentiate between one-dimensional and multi-dimensional arrays with examples.**

**Concept of Arrays in C**

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

- Each element can be accessed using an index (starting from 0).
- Arrays help store and manage multiple values under a single variable name

**Difference between one dimensional array and multi dimensional array:-**

| **Feature** | **one dimensional array** | **multi dimensional array** |
|---|---|---|
| Structure | single row of elements | table like(row,columns,etc) |
| Insdexing | one index[i] | multiple indices[i][j],[i][j][k] |
| Memory repress Antation | linear sequence | stored in row major order in memory |

| | | |
|---|---|---|
| Use case | storing simple list | matrics,table,grids |
| Example | int a[5]; | int a[3][4]; |

**Q2. 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.**

```
    // One-Dimensional Array
#include <stdio.h>
int main() {
    int arr1D[5];
    printf("Enter 5 integers:\n");
    for (int i = 0; i < 5; i++) {
        printf("Element %d: ", i + 1);
        scanf("%d", &arr1D[i]);
    }
    printf("\n1D Array elements:\n");
    for (int i = 0; i < 5; i++) {
        printf("%d ", arr1D[i]);
    }
    printf("\n");
    // Two-Dimensional Array (3x3 Matrix)
    int matrix[3][3], sum = 0;
    printf("\nEnter elements for 3x3 matrix:\n");
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            printf("Element [%d][%d]: ", i, j);
            scanf("%d", &matrix[i][j]);
            sum += matrix[i][j]; // Add to sum while reading
        }
    }
    printf("\nMatrix elements:\n");
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            printf("%d\t", matrix[i][j]);
        }
        printf("\n");
    }
    printf("\nSum of all matrix elements = %d\n", sum);

    return 0;
}
```

## 10. Pointers in C

**Q1. Explain what pointers are in C and how they are declared and initialized. Why are pointers important in C?**

A pointer is a variable that stores the memory address of another variable. Instead of holding actual data, a pointer holds where the data is located in memory.

**Pointer Declaration**

data_type *pointer_name;

- data_type → type of the data the pointer will point to (not the type of the pointer itself).

- * → indicates it's a pointer variable.

**Pointer initialization**

Pointer must store a valid address before use.

1. Assign address of variable using address of operator(&)
2. Assign memory dynamically using malloc (later topics).

**Pointers Important in C**

1. Efficient memory access – Directly work with memory locations.
2. Function arguments by reference – Modify variables in functions without returning them.
3. Dynamic memory allocation – Allocate memory at runtime (malloc, calloc, free).
4. Array and string manipulation – Arrays and pointers are closely related in C.
5. Building complex data structures – Like linked lists, trees, graphs.
6. Hardware-level programming – Useful in embedded systems, OS kernels.

**Q2. Write a C program to demonstrate pointer usage. Use a pointer to modify the value of a variable and print the result.**

```
#include <stdio.h>

int main() {

   int num = 10;

   int *ptr;

   ptr = &num;

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

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

   printf("Pointer ptr holds: %p\n", (void*)ptr);
```

```
    printf("Value at ptr (dereferenced): %d\n", *ptr);
  *ptr = 25;
    printf("\nAfter modifying through pointer:\n");
    printf("New value of num: %d\n", num);
    printf("Value at ptr: %d\n", *ptr);
    return 0;
}
```

**11. Strings in C**

**Q1. Explain string handling functions like strlen(), strcpy(), strcat(), strcmp(), and strchr(). Provide examples of when these functions are useful.**

**1. strlen() – String Length**

**Purpose:** Returns the length of a string **excluding** the null character '\0'.

**Example:**

```
#include <stdio.h>
#include <string.h>
int main() {
    char name[] = "Hello";
    printf("Length: %zu\n", strlen(name)); // Output: 5
    return 0;
}
```

**2. strcpy() – String Copy**

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

**Example:**

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

```
#include <string.h>

int main() {

   char src[] = "C Language";

   char dest[20];

   strcpy(dest, src);

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

   return 0;

}
```

**3. strcat() – String Concatenation**

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

**Example:**

```
#include <stdio.h>

#include <string.h>

int main() {

   char str1[20] = "Hello ";

   char str2[] = "World";

   strcat(str1, str2);

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

   return 0;

}
```

**4. strcmp() – String Compare**

**Purpose:** Compares two strings lexicographically.

- Returns **0** if strings are equal.
- Returns **>0** if first string is greater.
- Returns **<0** if first string is smaller.

**Example:**

```c
#include <stdio.h>
#include <string.h>
int main() {
    char s1[] = "apple";
    char s2[] = "banana";

    int result = strcmp(s1, s2);
    if (result == 0)
        printf("Strings are equal\n");
    else if (result < 0)
        printf("First string is smaller\n");
    else
        printf("First string is greater\n");

    return 0;
}
```

**5. strchr() – Character Search in String**

**Purpose:** Finds the first occurrence of a character in a string.
Returns a pointer to that character or NULL if not found.

**Example:**

```c
#include <stdio.h>
#include <string.h>
int main() {
    char text[] = "Programming";
    char *pos = strchr(text, 'g');

    if (pos != NULL)
        printf("Found 'g' at position: %ld\n", pos - text);
    else
        printf("Character not found\n");

    return 0;
}
```

**Q2. 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().**

```
#include <stdio.h>
#include <string.h>
int main() {
    char str1[100], str2[50];
    printf("Enter first string: ");
    fgets(str1, sizeof(str1), stdin);
    str1[strcspn(str1, "\n")] = '\0';
    printf("Enter second string: ");
    fgets(str2, sizeof(str2), stdin);
    str2[strcspn(str2, "\n")] = '\0';
    strcat(str1, str2);
    printf("\nConcatenated String: %s\n", str1);
    printf("Length of concatenated string: %zu\n", strlen(str1));
```

```
   return 0;
}
```

## 12. Structures in C

**Q1. Explain the concept of structures in C. Describe how to declare, initialize, and access structure members.**

A **structure** in C is a **user-defined data type** that allows you to group **different types of data** under a single name.

- It's like a container that can hold variables of different data types, called **members**.
- Useful for representing **real-world entities** where attributes have different types.

**Example:**
If you want to store information about a student — name (string), roll number (int), and marks (float) — you can keep them together using a structure.

**Declaring a Structure**

**Syntax:**

```
struct structure_name {
   data_type member1;
   data_type member2;
   ...
};
```

**Declaring Structure Variables**

After defining a structure, you can create variables of that type:

```
struct Student s1, s2;
```

Or declare variables at the time of definition:

```
struct Student {
    char name[50];
    int roll;
    float marks;
} s1, s2;
```

**Initializing a Structure**

You can initialize a structure in two ways:

**1. At declaration**

```
struct Student s1 = {"Alice", 101, 95.5};
```

**2. Assigning members individually**

```
struct Student s2;
strcpy(s2.name, "Bob");
s2.roll = 102;
s2.marks = 88.0;
```

**Accessing Structure Members**

- Use the **dot (.) operator** for normal variables.
- Use the **arrow (->) operator** for pointers to structures.

**Example:**

```
#include <stdio.h>
#include <string.h>
struct Student {
    char name[50];
    int roll;
    float marks;
};
int main() {
    struct Student s1;

    strcpy(s1.name, "Alice");

    s1.roll = 101;

    s1.marks = 95.5;

    printf("Name: %s\n", s1.name);

    printf("Roll: %d\n", s1.roll);

    printf("Marks: %.2f\n", s1.marks);

    return 0;

}
```

**Q2. 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.**

```
#include <stdio.h>

#include <string.h>

struct Student {

    char name[50];

    int roll;

    float marks;

};

int main() {

    struct Student students[3];

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

        printf("\nEnter details of student %d:\n", i + 1);

        printf("Name: ");

        scanf(" %[^\n]", students[i].name);

        printf("Roll Number: ");
        scanf("%d", &students[i].roll);
        printf("Marks: ");
        scanf("%f", &students[i].marks);
    }
    printf("\n--- Student 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].roll);
        printf("Marks: %.2f\n", students[i].marks);
    }
    return 0;
}
```

## 13. File Handling in C

**Q1. Explain the importance of file handling in C. Discuss how to perform file operations like opening, closing, reading, and writing files.**

**Importance of File Handling in C**

In C, variables store data **temporarily** in RAM, which is lost when the program ends.
**File handling** allows:

- **Permanent storage** of data.
- Reading existing data and processing it.
- Handling large datasets.
- Data exchange between programs.
- Logging program activities.

Without file handling, every time the program restarts, all data would have to be entered again.

## 2. File Handling Functions

All file handling functions are in the **<stdio.h>** library.

| **Function** | **Purpose** |
|---|---|
| fopen() | Opens a file. |
| fclose() | Closes a file. |
| fprintf() | Writes formatted data to a file. |
| fscanf() | Reads formatted data from a file. |
| fgets() | Reads a string from a file. |
| fputs() | Writes a string to a file. |
| fread() | Reads binary data from a file. |
| fwrite() | Writes binary data to a file. |

## 3. Opening a File

```
FILE *fp;

fp = fopen("data.txt", "w"); // "w" = write mode
```

**File Modes:**

- "r" → Read (file must exist)
- "w" → Write (creates new file or overwrites existing)
- "a" → Append (adds data to end of file)
- "r+" → Read and write
- "w+" → Read and write (overwrite file)

- "a+" → Read and append

**4. Closing a File**

```
fclose(fp);
```

Closes the file to prevent memory leaks and save data properly.

**5. Writing to a File**

```
#include <stdio.h>
int main() {
   FILE *fp = fopen("data.txt", "w");
   if (fp == NULL) {
      printf("Error opening file!\n");
      return 1;
   }
   fprintf(fp, "Hello, File Handling in C!\n");
   fclose(fp);
   return 0;
}
```

**6. Reading from a File**

```
#include <stdio.h>
int main() {
   char buffer[100];
   FILE *fp = fopen("data.txt", "r");
   if (fp == NULL) {
       printf("Error opening file!\n");
       return 1;
   }
   while (fgets(buffer, sizeof(buffer), fp) != NULL) {
       printf("%s", buffer);
   }
   fclose(fp);
   return 0;
}
```

**7. Steps to Perform File Handling in C**

1. **Declare a file pointer:**

```
FILE *fp;
```

2. **Open the file using fopen()** with correct mode.
3. **Check if the file opened successfully.**
4. **Read or write using appropriate functions.**
5. **Close the file using fclose().**

**Q2. 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.**

```
#include <stdio.h>
int main() {
    FILE *fp;
    char str[100] = "Hello, this is a file handling example in C.";
    char buffer[100];
    fp = fopen("example.txt", "w");
    if (fp == NULL) {
        printf("Error creating file!\n");
        return 1;
    }
    fprintf(fp, "%s", str);
    fclose(fp);
    printf("Data written to file successfully.\n");
    fp = fopen("example.txt", "r");
    if (fp == NULL) {
        printf("Error opening file for reading!\n");
        return 1;
    }
    printf("\nReading from file:\n");
    while (fgets(buffer, sizeof(buffer), fp) != NULL) {
        printf("%s", buffer);
    }
    fclose(fp);
    return 0;
}
```

## EXTRA LAB EXERCISES FOR IMPROVING PROGRAMMING LOGIC

**1. Operators**

**LAB EXERCISE 1: Simple Calculator**

- **Write a C program that acts as a simple calculator. The program should take two numbers and an operator as input from the user and perform the respective operation (addition, subtraction, multiplication, division, or modulus) using operators.**
- **Challenge: Extend the program to handle invalid operator inputs.**

```
#include <stdio.h>

int main() {

   double num1, num2;

   char operator;

   printf("Enter first number: ");

   scanf("%lf", &num1);

   printf("Enter an operator (+, -, *, /, %%): ");

   scanf(" %c", &operator);

   printf("Enter second number: ");

   scanf("%lf", &num2);

   switch (operator) {

      case '+':

         printf("Result: %.2lf\n", num1 + num2);

         break;

      case '-':

         printf("Result: %.2lf\n", num1 - num2);

         break;

      case '*':

         printf("Result: %.2lf\n", num1 * num2);

         break;
```

```
        case '/':
            if (num2 != 0)
                printf("Result: %.2lf\n", num1 / num2);
            else
                printf("Error: Division by zero is not allowed.\n");
            break;
        case '%':
            if ((int)num2 != 0)
                printf("Result: %d\n", (int)num1 % (int)num2);
            else
                printf("Error: Modulus by zero is not allowed.\n");
            break;
        default:
            printf("Error: Invalid operator '%c'. Please use +, -, *, /, or %%.\n",
operator);
    }
    return 0;
}
```

**LAB EXERCISE 2: Check Number Properties**

**• Write a C program that takes an integer from the user and checks the following using different operators: o Whether the number is even or odd. o Whether the number is positive, negative, or zero. o Whether the number is a multiple of both 3 and 5.**

```
#include <stdio.h>
int main() {
    int num;
    printf("Enter an integer: ");
    scanf("%d", &num);
    if (num % 2 == 0)
        printf("%d is Even.\n", num);
    else
        printf("%d is Odd.\n", num);
    if (num > 0)
        printf("%d is Positive.\n", num);
    else if (num < 0)
        printf("%d is Negative.\n", num);
    else
        printf("The number is Zero.\n");
    if (num % 3 == 0 && num % 5 == 0)
        printf("%d is a multiple of both 3 and 5.\n", num);
    else
        printf("%d is NOT a multiple of both 3 and 5.\n", num);
    return 0;
}
```

**2.Control Statements**

**LAB EXERCISE 1: Grade Calculator**

**• Write a C program that takes the marks of a student as input and displays the corresponding grade based on the following conditions: o Marks > 90: Grade A o Marks > 75 and <= 90: Grade B o Marks > 50 and <= 75: Grade C o**

**Marks <= 50: Grade D • Use if-else or switch statements for the decision-making process.**

```
#include <stdio.h>
int main() {
   float marks;
   printf("Enter the marks of the student: ");
   scanf("%f", &marks);
   if (marks > 90) {
      printf("Grade: A\n");
   }
   else if (marks > 75 && marks <= 90) {
      printf("Grade: B\n");
   }
   else if (marks > 50 && marks <= 75) {
      printf("Grade: C\n");
   }
   else {
      printf("Grade: D\n");
   }
   return 0;
}
```

**LAB EXERCISE 2: Number Comparison**

**• Write a C program that takes three numbers from the user and determines: o The largest number. o The smallest number. • Challenge: Solve the problem using both if-else and switch-case statements.**

```
#include <stdio.h>
int main() {
    int a, b, c;
    int largest, smallest;
    int choice;
    printf("Enter three numbers: ");
    scanf("%d %d %d", &a, &b, &c);
    if (a >= b && a >= c)
        largest = a;
    else if (b >= a && b >= c)
        largest = b;
    else
        largest = c;
    if (a <= b && a <= c)
        smallest = a;
    else if (b <= a && b <= c)
        smallest = b;
    else
        smallest = c;
    printf("\n[Using if-else]\n");
    printf("Largest: %d\n", largest);
    printf("Smallest: %d\n", smallest);
    printf("\nEnter 1 to find Largest or 2 to find Smallest (using switch): ");
    scanf("%d", &choice);
    switch (choice) {
        case 1: // Largest
            largest = (a >= b && a >= c) ? a : (b >= a && b >= c) ? b : c;
            printf("Largest: %d\n", largest);
            break;
        case 2:
            smallest = (a <= b && a <= c) ? a : (b <= a && b <= c) ? b : c;
            printf("Smallest: %d\n", smallest);
            break;
        default:
            printf("Invalid choice.\n");
    }
    return 0;
}
```

3.**Loops**

**LAB EXERCISE 1: Prime Number Check**

**Write a C program that checks whether a given number is a prime number or not using a for loop. • Challenge: Modify the program to print all prime numbers between 1 and a given number.**

```
#include <stdio.h>
int main() {
    int num, i, isPrime;
    printf("Enter a number to check if it is prime: ");
    scanf("%d", &num);

    if (num <= 1) {
        printf("%d is NOT a prime number.\n", num);
    } else {
        isPrime = 1; // assume prime
        for (i = 2; i <= num / 2; i++) {
            if (num % i == 0) {
                isPrime = 0;
                break;
            }
        }
        if (isPrime)
            printf("%d is a prime number.\n", num);
        else
            printf("%d is NOT a prime number.\n", num);
    }
    printf("\nPrime numbers between 1 and %d are:\n", num);
    for (int n = 2; n <= num; n++) {
        isPrime = 1;
        for (i = 2; i <= n / 2; i++) {
            if (n % i == 0) {
                isPrime = 0;
                break;
            }
        }
        if (isPrime)
            printf("%d ", n);
    }
    printf("\n");
    return 0;
}
```

**LAB EXERCISE 2: Multiplication Table**

**• Write a C program that takes an integer input from the user and prints its multiplication table using a for loop. • Challenge: Allow the user to input the range of the multiplication table (e.g., from 1 to N).**

```
#include <stdio.h>
int main() {
    int num, start, end;
    printf("Enter an integer to print its multiplication table: ");
    scanf("%d", &num);
    printf("Enter starting range: ");
    scanf("%d", &start);
    printf("Enter ending range: ");
    scanf("%d", &end);
    if (start > end) {
        printf("Invalid range! Starting value should be less than or equal to ending
value.\n");
        return 0;
    }

  printf("\nMultiplication Table of %d (from %d to %d):\n", num, start, end);
    for (int i = start; i <= end; i++) {
        printf("%d x %d = %d\n", num, i, num * i);
    }
    return 0;
}
```

**LAB EXERCISE 3: Sum of Digits**

**• Write a C program that takes an integer from the user and calculates the sum of its digits using a while loop. • Challenge: Extend the program to reverse the digits of the number.**

```
#include <stdio.h>
int main() {
    int num, temp, digit, sum = 0, reverse = 0;
    printf("Enter an integer: ");
    scanf("%d", &num);
      while (temp != 0) {
        digit = temp % 10;
        sum += digit;
        reverse = reverse * 10 + digit;
        temp /= 10;
    }
  printf("Sum of digits of %d = %d\n", num, sum);
    printf("Reversed number = %d\n", reverse);
 return 0;
}
```

**4.Arrays**

**• LAB EXERCISE 1: Maximum and Minimum in Array**

- **Write a C program that accepts 10 integers from the user and stores them in an array. The program should then find and print the maximum and minimum values in the array.Challenge: Extend the program to sort the array in ascending order.**

```
#include <stdio.h>

int main() {

   int n, i, j, temp;

printf("Enter number of elements: ");

   scanf("%d", &n);

   int arr[n];

   printf("Enter %d elements:\n", n);

   for (i = 0; i < n; i++) {

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

   }

   for (i = 0; i < n - 1; i++) {

      for (j = 0; j < n - i - 1; j++) {

         if (arr[j] > arr[j + 1]) {

               temp = arr[j];

    arr[j] = arr[j + 1];

            arr[j + 1] = temp;

         }

      }

   }

   printf("Array in ascending order:\n");

   for (i = 0; i < n; i++) {

      printf("%d ", arr[i]);
```

```
  }
   printf("\n");
  return 0;
}
```

**LAB EXERCISE 2: Matrix Addition**

**• Write a C program that accepts two 2x2 matrices from the user and adds them. Display the resultant matrix. Challenge: Extend the program to work with 3x3 matrices and matrix multiplication.**

```
#include <stdio.h>
int main() {
    int a[2][2], b[2][2], sum[2][2];
    int i, j;
 printf("Enter elements of first 2x2 matrix:\n");
    for(i = 0; i < 2; i++) {
        for(j = 0; j < 2; j++) {
            scanf("%d", &a[i][j]);
        }
 }
 printf("Enter elements of second 2x2 matrix:\n");
    for(i = 0; i < 2; i++) {
        for(j = 0; j < 2; j++) {
            scanf("%d", &b[i][j]);
        }
    }

  for(i = 0; i < 2; i++) {
        for(j = 0; j < 2; j++) {
            sum[i][j] = a[i][j] + b[i][j];
        }
    }
 printf("Resultant matrix after addition:\n");
    for(i = 0; i < 2; i++) {
        for(j = 0; j < 2; j++) {
            printf("%d\t", sum[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

**LAB EXERCISE 3: Sum of Array Elements**

**• Write a C program that takes N numbers from the user and stores them in an array. The program should then calculate and display the sum of all array elements. Challenge: Modify the program to also find the average of the numbers.**

```
#include <stdio.h>
int main() {
    int n, i;
    float sum = 0, average;
printf("Enter the number of elements: ");
    scanf("%d", &n);
 int arr[n];
```

```
    printf("Enter %d numbers:\n", n);
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
        sum += arr[i];
    }
    average = sum / n;
    printf("Sum of all elements: %.2f\n", sum);
    printf("Average of the elements: %.2f\n", average);
    return 0;
}
```

**5.Functions**

**LAB EXERCISE 1: Fibonacci Sequence**

**• Write a C program that generates the Fibonacci sequence up to N terms using a recursive function. • Challenge: Modify the program to calculate the Nth Fibonacci number using both iterative and recursive methods. Compare their efficiency.**

```
#include <stdio.h>
int fibonacci(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
int main() {
    int n, i;
    printf("Enter number of terms: ");
    scanf("%d", &n);
    printf("Fibonacci sequence up to %d terms:\n", n);
    for (i = 0; i < n; i++) {
        printf("%d ", fibonacci(i));
    }
    return 0;
}
```

**Find the Nth Fibonacci number using both iterative & recursive methods and compare**

```
#include <stdio.h>
#include <time.h>
int fibonacci_recursive(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2);
}
int fibonacci_iterative(int n) {
    int a = 0, b = 1, c, i;
    if (n == 0) return a;
    for (i = 2; i <= n; i++) {
        c = a + b;
        a = b;
        b = c;
    }
    return b;
}
int main() {
    int n;
    clock_t start, end;
    double time_taken;
    printf("Enter the value of N: ");
    scanf("%d", &n);
    start = clock();
    int fib_rec = fibonacci_recursive(n);
    end = clock();
    time_taken = ((double)(end - start))
    printf("Recursive: %dth Fibonacci number = %d (Time: %f seconds)\n", n,
fib_rec, time_taken);
    start = clock();
    int fib_itr = fibonacci_iterative(n);
    end = clock();
    time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
    printf("Iterative: %dth Fibonacci number = %d (Time: %f seconds)\n", n,
fib_itr, time_taken);

    return 0;
}
```

**LAB EXERCISE 2: Factorial Calculation**

**• Write a C program that calculates the factorial of a given number using a function. • Challenge: Implement both an iterative and a recursive version of the factorial function and compare their performance for large numbers.**

**Factorial using a function**

```
#include <stdio.h>
long long factorial(int n) {
    long long fact = 1;
    for (int i = 1; i <= n; i++) {
        fact *= i;
    }
    return fact;
}
int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);

    printf("Factorial of %d = %lld\n", num, factorial(num));
    return 0;
}
```

**Challenge – Iterative vs Recursive with Performance Comparison**

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

long long factorial_recursive(int n) {
    if (n == 0 || n == 1)
```

```
        return 1;
    else
        return n * factorial_recursive(n - 1);
}
long long factorial_iterative(int n) {
    long long fact = 1;
    for (int i = 1; i <= n; i++) {
        fact *= i;
    }
    return fact;
}
int main() {
    int n;
    clock_t start, end;
    double time_taken;

    printf("Enter a number: ");
    scanf("%d", &n);

    // Recursive approach
    start = clock();
    long long fact_rec = factorial_recursive(n);
    end = clock();
    time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
    printf("Recursive: Factorial of %d = %lld (Time: %f sec)\n", n, fact_rec,
time_taken);

    // Iterative approach
    start = clock();
    long long fact_itr = factorial_iterative(n);
    end = clock();
    time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
    printf("Iterative: Factorial of %d = %lld (Time: %f sec)\n", n, fact_itr,
time_taken);

    return 0;
}
```

**LAB EXERCISE 3: Palindrome Check**

**• Write a C program that takes a number as input and checks whether it is a palindrome using a function. • Challenge: Modify the program to check if a given string is a palindrome.**

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
int isPalindromeNumber(int num) {
    int original = num, reversed = 0, digit;
    while (num > 0) {
        digit = num % 10;
        reversed = reversed * 10 + digit;
        num /= 10;
    }
    return (original == reversed);
}
int isPalindromeString(char str[]) {
    int i = 0, j = strlen(str) - 1;
    while (i < j) {
        while (i < j && !isalnum(str[i])) i++;
        while (i < j && !isalnum(str[j])) j--;
        if (tolower(str[i]) != tolower(str[j]))
            return 0;
        i++;
        j--;
    }
    return 1;
}
int main() {
    int choice;
    printf("Choose option:\n1. Check Number Palindrome\n2. Check String
Palindrome\nEnter choice: ");
    scanf("%d", &choice);
    if (choice == 1) {
        int num;
        printf("Enter a number: ");
        scanf("%d", &num);
        if (isPalindromeNumber(num))
            printf("%d is a palindrome number.\n", num);
        else
            printf("%d is not a palindrome number.\n", num);
    }
    else if (choice == 2) {
        char str[100];
        printf("Enter a string: ");
        scanf(" %[^\n]", str);
        if (isPalindromeString(str))
            printf("\"%s\" is a palindrome string.\n", str);
        else
            printf("\"%s\" is not a palindrome string.\n", str);
    }
    else {
        printf("Invalid choice!\n");
    }
    return 0;
}
```

**6.Strings**

**LAB EXERCISE 1: String Reversal**

**• Write a C program that takes a string as input and reverses it using a function. • Challenge: Write the program without using built-in string handling functions.**

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

```c
int stringLength(char str[]) {
    int len = 0;
    while (str[len] != '\0') {
        len++;
    }
    return len;
}
void reverseString(char str[]) {
    int i, j;
    char temp;
    int len = stringLength(str);
    for (i = 0, j = len - 1; i < j; i++, j--) {
        temp = str[i];
        str[i] = str[j];
        str[j] = temp;
    }
}
int main() {
    char str[100];
    printf("Enter a string: ");
    scanf(" %[^\n]", str);
    reverseString(str);
    printf("Reversed string: %s\n", str);
    return 0;
}
```

**LAB EXERCISE 2: Count Vowels and Consonants**

**• Write a C program that takes a string from the user and counts the number of vowels and consonants in the string. • Challenge: Extend the program to also count digits and special characters.**

```
#include <stdio.h>

#include <ctype.h>

void countCharacters(char str[], int *vowels, int *consonants, int *digits, int
*specials) {

   int i = 0;

   *vowels = *consonants = *digits = *specials = 0;

   while (str[i] != '\0') {

      char ch = str[i];

      if (isalpha(ch)) {

         char lower = tolower(ch);

         if (lower == 'a' || lower == 'e' || lower == 'i' || lower == 'o' || lower ==
'u')

            (*vowels)++;

         else

            (*consonants)++;

      }

      else if (isdigit(ch)) {

         (*digits)++;

      }

      else if (ch != ' ') {

         (*specials)++;
```

```
        }
        i++;
    }
}

int main() {
    char str[200];
    int vowels, consonants, digits, specials;
    printf("Enter a string: ");
    scanf(" %[^\n]", str);
    countCharacters(str, &vowels, &consonants, &digits, &specials);
    printf("\nVowels: %d\n", vowels);
    printf("Consonants: %d\n", consonants);
    printf("Digits: %d\n", digits);
    printf("Special characters: %d\n", specials);
    return 0;
}
```

**LAB EXERCISE 3: Word Count**

**• Write a C program that counts the number of words in a sentence entered by the user. • Challenge: Modify the program to find the longest word in the sentence.**

```
#include <stdio.h>
int countWords(char str[]) {
    int i = 0, count = 0, inWord = 0;
    while (str[i] != '\0') {
        if (str[i] != ' ' && str[i] != '\t' && str[i] != '\n') {
            if (inWord == 0) {
                inWord = 1;
                count++;
            }
        } else {
            inWord = 0;
        }
        i++;
    }
    return count;
}
void findLongestWord(char str[], char longest[]) {
    int i = 0, start = 0, len = 0;
    int maxLen = 0, maxStart = 0;
    while (1) {
        if (str[i] == ' ' || str[i] == '\0') {
            if (len > maxLen) {
                maxLen = len;
                maxStart = start;
            }
            if (str[i] == '\0')
                break;
            len = 0;
            start = i + 1;
        } else {
            len++;
        }
        i++;
    }
    for (i = 0; i < maxLen; i++) {
        longest[i] = str[maxStart + i];
    }
    longest[maxLen] = '\0';
}
int main() {
    char str[200], longest[50];
    printf("Enter a sentence: ");
    scanf(" %[^\n]", str);
    int words = countWords(str);
    findLongestWord(str, longest);
    printf("Number of words = %d\n", words);
    printf("Longest word = %s\n", longest);
    return 0;
}
```