Module 2 – Introduction to Programming

Overview of C Programming

- **THEORY EXERCISE:**

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

Ans. The C programming language has a rich history that spans over four decades. Developed by Dennis Ritchie between 1969 and 1973, C was designed to be a portable, efficient, and easy-to-use language for writing operating systems and other low-level software.

In the early 1970s, computers were large, expensive, and limited in their capabilities. Programming languages at the time, such as COBOL and FORTRAN, were designed for specific tasks and were not well-suited for systems programming. Ritchie, who worked at Bell Labs, recognized the need for a language that could be used to write operating systems, device drivers, and other low-level software.

C was first implemented on the PDP-11 minicomputer and was initially called "NB" (New B). The language was later renamed to C, and its first compiler was written in assembly language. C quickly gained popularity due to its efficiency, portability, and flexibility.

Throughout the 1970s and 1980s, C underwent several revisions and expansions. In 1978, Ritchie and Brian Kernighan published "The C Programming Language," a book that became the de facto standard for the language. The book introduced the concept of "K&R C," which became the basis for the ANSI C standard in 1989.

The importance of C cannot be overstated. It has had a profound impact on the development of modern computing. C's efficiency, portability, and flexibility made it an ideal choice for writing operating systems, device drivers, and other low-level software. Many operating systems, including Unix and Linux, were written in C.

C's influence extends beyond operating systems. It has been used to write a wide range of applications, including web browsers, databases, and games. Many programming languages, including C++, Java, and Python, have been influenced by C's syntax and semantics.

Despite the rise of newer programming languages, C remains widely used today. Its importance can be attributed to several factors:

1. Performance: C's compilation to native machine code makes it one of the fastest programming languages available.

2. Portability: C's standard library and lack of platform-specific features make it easy to write cross-platform code.

3. Reliability: C's lack of runtime checks and errors makes it a popular choice for systems programming and embedded systems.

4. Flexibility: C's low-level memory management and lack of object-oriented features make it a versatile language for a wide range of applications.

● **LAB EXERCISE:**

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

Ans. Embedded Systems: Traffic Management Systems

C programming is widely used in embedded systems, such as traffic management systems. These systems use sensors, cameras, and other devices to monitor and control traffic flow. C is used to write the firmware for these devices, which requires low-level memory management, efficient code execution, and reliability.

For example, the city of Singapore uses a traffic management system called the "Intelligent Transport System" (ITS), which uses C programming to manage traffic flow, monitor traffic conditions, and optimize traffic signal timing.

Operating Systems : Linux Kernel

C programming is used extensively in operating system development, particularly in the Linux kernel. The Linux kernel is written mostly in C, with some parts written in assembly language. C is used to write device drivers, file systems, and other low-level system components.

The Linux kernel is used in a wide range of devices, from smartphones to supercomputers. Its reliability, efficiency, and customizability make it a popular choice for many applications.

Game Development : Game Engines

C programming is also used in game development, particularly in game engines. Game engines provide a framework for building games, and they often require low-level memory management, efficient code execution, and reliability.

For example, the id Tech 3 game engine, used in games such as Quake III Arena, is written mostly in C. The engine provides a high-performance, cross-platform framework for building games, and C is used to write the engine's core components, such as the rendering engine and the physics engine.

2. Setting Up Environment

● THEORY EXERCISE:

o 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.

Ans. Installing a C Compiler (GCC)

1. Windows:

- Download the MinGW installer from the official website: https://www.mingw-w64.org/downloads/

- Follow the installation instructions to install MinGW, which includes the GCC compiler.

- Add the MinGW bin directory to your system's PATH environment variable.

2. Linux (Ubuntu-based):

- Open a terminal and run the command: sudo apt-get install gcc

- Verify the installation by running the command: gcc --version

3. macOS (with Homebrew):

- Install Homebrew by following the instructions on the official website: https://brew.sh/

- Run the command: brew install gcc

- Verify the installation by running the command: gcc --version

Setting up an Integrated Development Environment (IDE) :

Option 1: DevC++

1. Download the DevC++ installer from the official website: https://www.bloodshed.net/

2. Follow the installation instructions to install DevC++.

3. Launch DevC++ and create a new project.

4. Configure the project settings to use the GCC compiler.

Option 2: VS Code

1. Download the VS Code installer from the official website: https://code.visualstudio.com/

2. Follow the installation instructions to install VS Code.

3. Launch VS Code and install the C/C++ extension by searching for "C/C++" in the Extensions panel.

4. Create a new file with a .c extension and start writing your C code.

5. Configure the compiler settings by creating a tasks.json file and specifying the GCC compiler.

Option 3: CodeBlocks

1. Download the CodeBlocks installer from the official website: https://www.codeblocks.org/

2. Follow the installation instructions to install CodeBlocks.

3. Launch CodeBlocks and create a new project.

4. Configure the project settings to use the GCC compiler.

● **LAB EXERCISE:**

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

Ans. Installing a C Compiler

Windows :

1. Download the MinGW installer from the official website: https://www.mingw-w64.org/downloads/

2. Follow the installation instructions to install MinGW, which includes the GCC compiler.

3. Add the MinGW bin directory to your system's PATH environment variable.

Configuring an IDE :

For this example, we'll use VS Code as our IDE.

Installing VS Code

1. Download the VS Code installer from the official website: https://code.visualstudio.com/

2. Follow the installation instructions to install VS Code.

Installing the C/C++ Extension :

1. Launch VS Code and open the Extensions panel by clicking the Extensions icon in the left sidebar or pressing Ctrl + Shift + X.

2. Search for "C/C++" in the Extensions panel.

3. Click the Install button to install the C/C++ extension.

Writing Your First C Program:

Creating a New File:

1. Launch VS Code and create a new file by clicking the New File button in the Explorer panel or pressing Ctrl + N.

2. Save the file with a .c extension, for example, hello.c.

Writing the Code:

1. Copy and paste the following code into the hello.c file:

```
#include <stdio.h>
int main() {
   printf("Hello, World!\n");
   return 0;
}
```

2. Save the file by clicking the Save button in the top-right corner or pressing Ctrl + S.

Compiling and Running the Code

1. Open a terminal in VS Code by clicking the Terminal icon in the top menu bar or pressing Ctrl +.

2. Navigate to the directory where you saved the hello.c file using the cd command.

3. Compile the code using the GCC compiler by running the command: gcc hello.c -o hello

Created by Gajendra Suthar.

4. Run the compiled program by running the command: ./hello

You should see the output "Hello, World!" printed to the terminal.

3. Basic Structure of a C Program

● THEORY EXERCISE:

o 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

Headers

Headers are files that contain function declarations, macro definitions, and other definitions that can be used by multiple source files. In C, headers typically have a .h extension.

```
#include <stdio.h>  // Standard input/output header
#include <stdlib.h>  // Standard library header
```

Main Function:

The main function is the entry point of a C program. It is where the program starts execution.

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

Comments:

Comments are used to add notes or explanations to the code. In C, comments start with /* and end with */, or start with // for single-line comments.

```
/*  a multi-line comment */
//  a single-line comment
```

Data Types:

C has several built-in data types, including:

- Integers (int)
- Characters (char)
- Floating-point numbers (float, double, long double)
- Boolean values (_Bool in C99 and later)

```
int x = 10;         // Integer variable
char c = 'A';       // Character variable
float f = 3.14;     // Floating-point variable
```

Variables:

Variables are used to store values. In C, variables must be declared before they can be used.

```
int x;              // Declare an integer variable
x = 10;             // Assign a value to the variable
```

Example Program:

Here's an example C program that demonstrates the basic structure:

```
#include <stdio.h>
int main() {
   /* Declare and initialize variables */
   int x = 10;
   char c = 'A';
   float f = 3.14;
   /* Print the values of the variables */
   printf("x = %d\n", x);
```

```
    printf("c = %c\n", c);
    printf("f = %f\n", f);
    return 0;
}
```

● LAB EXERCISE:

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

Ans.

```
#include <stdio.h>
int main() {
    // Declare and initialize integer variables
    int x = 10;      // decimal integer
    // Declare and initialize character variables
    char c1 = 'A';   // character constant
    char c2 = 65;    // ASCII value of 'A'
    // Declare and initialize floating-point variables
    float f1 = 3.14; // floating-point constant
    // Declare and initialize constants
    const int MAX_SIZE = 100; // constant integer
    const float PI = 3.14159; // constant floating-point

    // Display the values of the variables
    printf("Integer Variables:");
    printf("x = %d", x);
    printf("\nCharacter Variables:");
    printf("c1 = %c", c1);
    printf("c2 = %c", c2);
    printf("\nFloating-Point Variables:");
```

```
printf("f1 = %f", f1);
printf("\nConstants:");
printf("MAX_SIZE = %d", MAX_SIZE);
printf("PI = %f", PI);
}
```

4. Operators in C

- **THEORY EXERCISE:**

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

Ans. Arithmetic Operators :

Arithmetic operators are used to perform mathematical operations.

- + (addition)

- - (subtraction)

- * (multiplication)

- / (division)

- % (modulus, remainder of division)

- ++ (increment, increases the value by 1)

- -- (decrement, decreases the value by 1)

Example: int x = 5; int y = 3; int result = x + y;

Relational Operators :

Relational operators are used to compare values.

- == (equal to)
- != (not equal to)
- (greater than)
- < (less than)
- >= (greater than or equal to)
- <= (less than or equal to)

Example: int x = 5; if (x > 3) printf("x is greater than 3");

Logical Operators :

Logical operators are used to combine relational expressions.

- && (logical and)
- || (logical or)
- ! (logical not)

Example: int x = 5; if (x > 3 && x < 10) printf("x is between 3 and 10");

Assignment Operators :

Assignment operators are used to assign values to variables.

- = (assignment)
- += (addition assignment)
- -= (subtraction assignment)
- *= (multiplication assignment)
- /= (division assignment)
- %= (modulus assignment)

Example: int x = 5; x += 3; printf("x = %d", x);

Increment/Decrement Operators :

Increment/decrement operators are used to increase or decrease the value of a variable.

- ++ (increment)
- (decrement)

Example: int x = 5; x++; printf("x = %d", x);

Bitwise Operators :

Bitwise operators are used to perform operations on bits.

- & (bitwise and)
- | (bitwise or)
- ^ (bitwise xor)
- ~ (bitwise not)
- << (left shift)
- >> (right shift)

Example: int x = 5; int y = x << 1; printf("y = %d", y);

Conditional Operators :

Conditional operators are used to evaluate conditions and return values.

• LAB EXERCISE:

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

Ans.

```
#include <stdio.h>
main() {
  int num1, num2;
// Accept two integers from the user
  printf("Enter first integer: ");
  scanf("%d", &num1);
  printf("Enter second integer: ");
  scanf("%d", &num2);

  // Display arithmetic results
  printf("\nArithmetic Results:\n");
  printf(" Addition: %d\n", num1 + num2);
```

```
    printf("Subtraction: %d\n", num1 - num2);
    printf("Product: %d\n", num1 * num2);
    printf("Divide: %d\n", num1 / num2);
    printf("Modulus: %d\n",  num1 % num2);

    // Display relational results
    printf("\nRelational Results:\n");
    printf("Is Equal: %d\n", num1 == num2);
    printf("Is Not Equal: %d\n", num1 != num2);
    printf("Is Greater: %d\n", num1 > num2);
    printf("Is Less: %d\n", num1 < num2);
    printf("Is Greater or Equal: %d\n", num1 >= num2);
    printf("Is Less or Equal: %d\n", num1 <= num2);

    // Display logical results
    printf("\nLogical Results:");
    printf("AND Result: %d\n",  (num1 > 0) && (num2 > 0));
    printf("OR Result: %d\n",  (num1 > 0) || (num2 > 0));
    printf("NOT Result: %d\n",  !(num1 > 0));
}
```

5. Control Flow Statements in C

- **THEORY EXERCISE:**

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

Ans.

If Statement :

The if statement is used to execute a block of code if a certain condition is true.

```
if (condition) {
   // code to be executed
}
```

Example:

```
#include <stdio.h>
 main() {
   int x = 10;
   if (x > 5) {
      printf("x is greater than 5");
   }
}
```

If-Else Statement :

The if-else statement is used to execute a block of code if a certain condition is true, and another block of code if the condition is false.

```
if (condition) {
   // code to be executed if condition is true
} else {
   // code to be executed if condition is false
}
```

Example:

```
#include <stdio.h>
 main() {
   int x = 10;
   if (x > 5) {
      printf("x is greater than 5\n");
   } else {
```

```
        printf("x is less than or equal to 5\n");
    }
}
```

Nested If-Else Statement :

The nested if-else statement is used to execute a block of code if a certain condition is true, and another block of code if the condition is false, and also to check another condition if the first condition is true.

```
if (condition1) {
    if (condition2) {
        // code to be executed if both conditions are true
    } else {
        // code to be executed if condition1 is true and condition2 is false
    }
} else {
    // code to be executed if condition1 is false
}
```

Example:

```
#include <stdio.h>
 main() {
    int x = 10;
    int y = 5;
    if (x > 5) {
        if (y > 5) {
            printf("Both x and y are greater than 5\n");
        } else {
            printf("x is greater than 5, but y is not\n");
        }
```

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

Switch Statement :

The switch statement is used to execute a block of code based on the value of a variable.

```
switch (expression) {
    case value1:
        // code to be executed if expression equals value1
        break;
    case value2:
        // code to be executed if expression equals value2
        break;
    default:
        // code to be executed if expression does not equal any of the values
        break;
}
```

Example:

```
#include <stdio.h>
 main() {
    int day = 3;
    switch (day) {
        case 1:
            printf("Monday\n");
            break;
        case 2:
```

```
            printf("Tuesday\n");
            break;
        case 3:
            printf("Wednesday\n");
            break;
        default
            printf("Invalid day\n");
            break;
    }
}
```

● **LAB EXERCISE:**

o 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.).

Ans.

```
#include <stdio.h>
 main() {
    int num, month;
    printf("Enter a number: ");
    scanf("%d", &num);
     if (num % 2 == 0) {
        printf("%d is even\n", num);
    } else {
        printf("%d is odd\n", num);
    }
    printf("Enter a month number (1-12): ");
    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\n");
            break;
    }
}
```

6. Looping in C

- **THEORY EXERCISE:**

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

Ans.

Loop Differences :

- Condition Check: While loops and for loops check the condition before executing the body. Do-while loops check the condition after executing the body.

- Body Execution: While loops and for loops may not execute the body at all if the initial condition is false. Do-while loops always execute the body at least once.

- Loop Control: For loops provide a concise way to initialize, check, and increment a loop variable. While loops and do-while loops require separate statements for these tasks.

Choosing the Right Loop :

- While Loops: Use when the number of iterations is unknown or depends on a condition that changes within the loop.

- For Loops: Use when the number of iterations is fixed or can be calculated before the loop starts. Ideal for iterating over arrays or performing a task a specific number of times.

- Do-While Loops: Use when the loop body needs to execute at least once, regardless of the initial condition. Commonly used for input validation or when the loop body has side effects.

Example Use Cases :

- While Loop: Reading input from a file until the end is reached.

- For Loop: Iterating over an array of fixed size to perform calculations.

- Do-While Loop: Validating user input, ensuring that the loop body executes at least once to prompt the user.

- **LAB EXERCISE:**

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

Ans.

```
#include <stdio.h>
   main() {
   // Using while loop
   printf("Using while loop:\n");
```

```
    int i = 1;
    while (i <= 10) {
        printf("%d ", i);
        i++;
    }
    printf("\n");

    // Using for loop
    printf("Using for loop:\n");
    for (int j = 1; j <= 10; j++) {
        printf("%d ", j);
    }
    printf("\n");

    // Using do-while loop
    printf("Using do-while loop:\n");
    int k = 1;
    do {
        printf("%d ", k);
        k++;
    } while (k <= 10);
    printf("\n");
}
```

Here's the output of the program:

Using while loop:

1 2 3 4 5 6 7 8 9 10

Using for 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. Loop Control Statements

- **THEORY EXERCISE:**

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

Ans.

Break Statement :

The break statement is used to terminate the execution of a loop or a switch statement. When a break statement is encountered, the program control is transferred to the statement immediately following the loop or switch statement.

Example:

```
#include <stdio.h>
  main() {
   for (int i = 1; i <= 5; i++) {
      if (i == 3) {
         break;
      }
      printf("%d ", i);
   }
```

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

Output:

1 2

In this example, the break statement is used to terminate the loop when i equals 3.

Continue Statement :

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

Example:

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

Output:

1 2 4 5

In this example, the continue statement is used to skip the current iteration when i equals 3.

Goto Statement :

The goto statement is used to transfer program control to a labeled statement. The labeled statement is identified by a unique label followed by a colon (:).

Example:

```
#include <stdio.h>
   main() {
   int i = 1;
   printf("Before goto\n");
   goto label;
   printf("This statement will not be executed\n");
label:
   printf("After goto\n");
   return 0;
}
```

Output:

Before goto

After goto

- **LAB EXERCISE:**

o 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.

```
#include <stdio.h>
   main() {
   for (int i = 1; i <= 10; i++) {
      if (i == 5) {
         printf("Stopping at %d", i);
         break;
      }
      if (i == 3) {
         printf("Skipping %d", i);
         continue;
      }
      printf("%d ", i);
   }
   printf("\n");
   return 0;
}
```

Here's the output of the program:

1 2 Skipping 3

4 Stopping at 5

8. Functions in C

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

Ans.

What are Functions in C?

In C, a function is a block of code that performs a specific task. Functions are used to organize code, reduce repetition, and make programs more modular and reusable.

Function Declaration :

A function declaration, also known as a function prototype, is a statement that specifies the function's name, return type, and parameter list. The function declaration is usually placed at the top of the program or in a header file.

return-type function-name(parameter-list);

Example:

int add(int, int);

This declares a function named add that takes two int parameters and returns an int value.

Function Definition :

A function definition is the actual implementation of the function. It includes the function declaration and the function body.

```
return-type function-name(parameter-list) {
   // function body
}
```

Example:

```
int add(int x, int y) {
   return x + y;
}
```

This defines the add function, which takes two int parameters x and y, and returns their sum.

Calling a Function :

To call a function, simply use the function name followed by the required arguments in parentheses.

function-name(argument-list);

Example:

```
int result = add(5, 3);
printf("%d", result);  // Output: 8
```

This calls the add function with arguments 5 and 3, and stores the result in the result variable.

Example Program :

Here's an example program that demonstrates function declaration, definition, and calling:

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

```
int add(int, int);
int main() {
   int x = 5;
   int y = 3;
   int result = add(x, y);
   printf("%d + %d = %d", x, y, result);
   return 0;
}

// Function definition
int add(int x, int y) {
   return x + y;
}
```

- **LAB EXERCISE:**

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

Ans.

```
#include <stdio.h>
// Function declaration
long factorial(int);
int main() {
   int num;
   printf("Enter a positive integer: ");
   scanf("%d", &num);
```

```
    // Check if the input is a positive integer
    if (num < 0) {
        printf("Error: Factorial is not defined for negative numbers.\n");
    } else {
        // Function call
        long result = factorial(num);
        printf("Factorial of %d = %ld\n", num, result);
    }
}
// Function definition
long factorial(int n) {
    long fact = 1;
    for (int i = 1; i <= n; i++) {
        fact *= i;
    }
    return fact;
}
```

9. Arrays in C

- **THEORY EXERCISE:**

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

Ans.

What are Arrays in C?

In C, an array is a collection of elements of the same data type stored in contiguous memory locations. Each element is identified by an index or subscript that allows you to access and manipulate the element.

One-Dimensional Arrays :

A one-dimensional array is a linear collection of elements. Here's an example:

```
int scores[5] = {90, 80, 70, 60, 50};
```

In this example, scores is a one-dimensional array of integers with 5 elements. The elements are stored in contiguous memory locations, and each element can be accessed using its index.

For example, scores[0] refers to the first element (90), scores[1] refers to the second element (80), and so on.

Multi-Dimensional Arrays :

A multi-dimensional array is an array of arrays. Here's an example:

```
int matrix[2][3] = {
   {1, 2, 3},
   {4, 5, 6}
};
```

In this example, matrix is a two-dimensional array of integers with 2 rows and 3 columns. The elements are stored in contiguous memory locations, and each element can be accessed using its row and column indices.

For example, matrix[0][0] refers to the first element (1), matrix[0][1] refers to the second element (2), and so on.

Key Differences :

Here are the key differences between one-dimensional and multi-dimensional arrays:

- Structure: One-dimensional arrays are linear collections of elements, while multi-dimensional arrays are arrays of arrays.

- Indexing: One-dimensional arrays use a single index to access elements, while multi-dimensional arrays use multiple indices (one for each dimension).

- Memory Layout: One-dimensional arrays store elements in contiguous memory locations, while multi-dimensional arrays store elements in contiguous memory locations, but with each row or column stored contiguously.

- **LAB EXERCISE:**

o 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.

```
#include <stdio.h>
 main() {
   // One-dimensional array
   int oneDArray[5] = {1, 2, 3, 4, 5};
   printf("One-dimensional array elements:\n");
   for (int i = 0; i < 5; i++) {
```

```
        printf("%d ", oneDArray[i]);
    }
    printf("\n\n");
    // Two-dimensional array (3x3 matrix)
    int twoDArray[3][3] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };
    printf("Two-dimensional array (3x3 matrix) elements:\n");
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            printf("%d ", twoDArray[i][j]);
        }
        printf("\n");
    }

    // Calculate the sum of all elements in the 2D array
    int sum = 0;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            sum += twoDArray[i][j];
        }
    }
    printf("\nSum of all elements in the 2D array: %d\n", sum);
```

}

Here's the output:

One-dimensional array elements:

1 2 3 4 5

Two-dimensional array (3x3 matrix) elements:

1 2 3

4 5 6

7 8 9

Sum of all elements in the 2D array: 45

10. Pointers in C

- **THEORY EXERCISE:**

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

Ans.

What are Pointers in C?

In C, a pointer is a variable that stores the memory address of another variable. Pointers are used to indirectly access and manipulate the values stored in memory.

Declaring Pointers :

To declare a pointer, you use the asterisk symbol (*) before the pointer name. The syntax for declaring a pointer is:

data_type *pointer_name;

For example:

int *ptr;

This declares a pointer named ptr that points to an int variable.

Initializing Pointers :

To initialize a pointer, you can assign it the address of an existing variable using the address-of operator (&). The syntax for initializing a pointer is:

pointer_name = &variable_name;

For example:

int x = 10;

int *ptr = &x;

This initializes the pointer ptr with the address of the variable x.

Why are Pointers Important in C?

Pointers are important in C for several reasons:

1. Memory Management: Pointers allow you to manually manage memory, which is essential for efficient programming.

2. Dynamic Memory Allocation: Pointers are used to allocate memory dynamically using functions like malloc() and calloc().

3. Data Structures: Pointers are used to implement complex data structures like linked lists, trees, and graphs.

4. Function Arguments: Pointers are used to pass variables by reference to functions, allowing the function to modify the original variable.

5. Efficient Code: Pointers can be used to write efficient code by reducing the number of copies of data and improving memory access.

Example :

Here's an example code that demonstrates the use of pointers:

```
#include <stdio.h>
 main() {
   int x = 10;
   int *ptr = &x;
   printf("Value of x: %d", x);
   printf("Address of x: %p", (void *)&x);
   printf("Value of ptr: %p", (void *)ptr);
   printf("Value at ptr: %d", *ptr);
   *ptr = 20;
   printf("Value of x after modification: %d\n", x);
}
```

- **LAB EXERCISE:**

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

Ans.

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

```
main() {
   int x = 10;
   int *ptr = &x;
   printf("Initial value of x: %d\n", x);
   *ptr = 20;
   printf("Modified value of x: %d\n", x);
   printf("Address of x: %p\n", (void *)&x);
   printf("Value of ptr: %p\n", (void *)ptr);}
```

Here's the output:

Initial value of x: 10

Modified value of x: 20

Address of x: 0x7ffd85721a4

Value of ptr: 0x7ffd85721a4

11. Strings in C

● THEORY EXERCISE:

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

Ans.

strlen() :

The strlen() function calculates the length of a string, excluding the null terminator (\0).

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

```
    int length = strlen(str);
    printf("Length of string: %d\n", length);
    return 0;
}
```

Useful when: You need to know the length of a string, such as when allocating memory for a copy of the string.

strcpy() :

The strcpy() function copies the contents of one string to another.

```
#include <string.h>
main() {
    char src[] = "Hello, World!";
    char dest[20];
    strcpy(dest, src);
    printf("Copied string: %s\n", dest);
    return 0;
}
```

Useful when: You need to create a copy of a string, such as when modifying a string that should remain unchanged.

strcat() :

The strcat() function appends the contents of one string to another.

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

Useful when: You need to combine multiple strings into a single string.

strcmp() :

The strcmp() function compares two strings and returns an integer indicating their relationship.

```
#include <string.h>
main() {
   char str1[] = "Hello";
   char str2[] = "World";
   int result = strcmp(str1, str2);
   if (result < 0) {
      printf("%s is less than %s\n", str1, str2);
   } else if (result > 0) {
      printf("%s is greater than %s\n", str1, str2);
   } else {
      printf("%s is equal to %s\n", str1, str2);
   }
```

```
}
```

Useful when: You need to compare two strings, such as when sorting or searching for specific strings.

strchr() :

The strchr() function searches for the first occurrence of a character in a string and returns a pointer to that character.

```
#include <string.h>
main() {
   char str[] = "Hello, World!";
   char *ptr = strchr(str, ',');
   if (ptr != NULL) {
      printf("Found comma at position %ld\n", ptr - str);
   } else {
      printf("Comma not found\n");
   }
}
```

- **LAB EXERCISE:**

o 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.

```
#include <stdio.h>
#include <string.h>
main() {
   char str1[50];
   char str2[50];
   char concatenated_str[100];

   printf("Enter the first string: ");
   fgets(str1, sizeof(str1), stdin);
   str1[strcspn(str1, "\n")] = 0;

   printf("Enter the second string: ");
   fgets(str2, sizeof(str2), stdin);
   str2[strcspn(str2, "\n")] = 0; // Remove the newline character

   // Concatenate the two strings
   strcpy(concatenated_str, str1);
   strcat(concatenated_str, str2);

   // Display the concatenated string
   printf("Concatenated string: %s\n", concatenated_str);

   // Display the length of the concatenated string
   printf("Length of concatenated string: %lu\n", strlen(concatenated_str));
```

}

12. Structures in C

• THEORY EXERCISE:

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

Ans.

What are Structures in C?

In C, a structure (also known as a struct) is a collection of variables of different data types that are stored together in memory. Structures are used to represent complex data types, such as a point in 2D space, a rectangle, or a person's details.

Declaring Structures :

To declare a structure, you use the struct keyword followed by the name of the structure and the members of the structure enclosed in curly brackets. Here's an example:

```
struct Person {
   int age;
   char name[20];
   float height;
};
```

This declares a structure named Person with three members: age, name, and height.

Initializing Structures :

There are two ways to initialize structures in C:

Method 1: Initialization at Declaration

You can initialize a structure when you declare it. Here's an example:

```
struct Person {
    int age;
    char name[20];
    float height;
} person1 = {25, "John Doe", 1.75};
```

Method 2: Initialization Using Dot Notation

You can also initialize a structure using dot notation. Here's an example:

```
struct Person person1;
person1.age = 25;
strcpy(person1.name, "John Doe");
person1.height = 1.75;
```

Accessing Structure Members :

To access a structure member, you use the dot notation. Here's an example:

```
struct Person person1 = {25, "John Doe", 1.75};

printf("Name: %s\n", person1.name);

printf("Age: %d\n", person1.age);

printf("Height: %.2f\n", person1.height);
```

Array of Structures :

You can also declare an array of structures. Here's an example:

```
struct Person persons[2] = {
   {25, "John Doe", 1.75},
   {30, "Jane Doe", 1.60}
};
```

You can access the members of each structure in the array using the array index and dot notation. Here's an example:

```
printf("Name: %s\n", persons[0].name);

printf("Age: %d\n", persons[1].age);
```

- **LAB EXERCISE:**

o 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.

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

```
#include <string.h>
struct Student {
    char name[50];
    int rollNumber;
    float marks;
};
int main() {
    struct Student students[3];
    for (int i = 0; i < 3; i++) {
        printf("Enter details of student %d:\n", i + 1);
        printf("Name: ");
        fgets(students[i].name, sizeof(students[i].name), stdin);
        students[i].name[strcspn(students[i].name, "\n")] = 0;
        printf("Roll Number: ");
        scanf("%d", &students[i].rollNumber);
        printf("Marks: ");
        scanf("%f", &students[i].marks);
        getchar(); // Consume the newline character left in the input buffer
    }
    printf("\nDetails of 3 students:\n");
    for (int i = 0; i < 3; i++) {
        printf("Student %d: ", i + 1);
        printf("Name: %s", students[i].name);
        printf("Roll Number: %d", students[i].rollNumber);
        printf("Marks: %.2f", students[i].marks);
```

```
    }
}
```

13. File Handling in C

- **THEORY EXERCISE:**

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

Ans. Importance of File Handling in C

File handling is an essential aspect of C programming, allowing you to store and retrieve data from files. Here are some reasons why file handling is important in C:

1. Data Persistence: File handling enables you to store data in a persistent manner, even after the program terminates.

2. Data Sharing: Files can be shared between different programs, allowing for data exchange and collaboration.

3. Efficient Data Storage: Files provide an efficient way to store large amounts of data, reducing memory usage and improving program performance.

File Operations in C :

Opening a File

To open a file in C, you use the fopen() function, which returns a pointer to the file stream:

```
FILE *fp = fopen("filename.txt", "mode");
```

The mode parameter specifies the file access mode:

- "r": Open for reading (default)
- "w": Open for writing (truncate if exists)
- "a": Open for appending (create if doesn't exist)
- "r+": Open for reading and writing
- "w+": Open for reading and writing (truncate if exists)
- "a+": Open for reading and appending (create if doesn't exist)

Closing a File

To close a file in C, you use the fclose() function:

```
fclose(fp);
```

Reading from a File :

To read from a file in C, you can use various functions:

- fgetc() : Reads a single character
- fgets() : Reads a line of text
- fscanf() : Reads formatted input
- fread() : Reads binary data

Example using fgets():

```
char buffer[1024];
fgets(buffer, sizeof(buffer), fp);
```

Writing to a File :

To write to a file in C, you can use various functions:

- fputc() : Writes a single character

- fputs() : Writes a line of text

- fprintf() : Writes formatted output

- fwrite() : Writes binary data

Example using fprintf():

fprintf(fp, "Hello, World!\n");

- **LAB EXERCISE:**

o 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.

```
#include <stdio.h>
#include <stdlib.h>
main() {
   FILE *fp = fopen("example.txt", "w");
   if (fp == NULL) {
      perror("Error creating file");
      exit(1);
   }
   char *str = "Hello, World!";
   fprintf(fp, "%s\n", str);
   fclose(fp);
   fp = fopen("example.txt", "r");
   if (fp == NULL) {
```

```
    perror("Error opening file");
    exit(1);
  }
  char buffer[1024];
  while (fgets(buffer, sizeof(buffer), fp) != NULL) {
    printf("%s", buffer);
  }
  fclose(fp);}
```

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.**

Ans.

```
#include<stdio.h>
main()
{
    int a, b, choice;
    printf("\n\n\t enter any two number :");
    scanf("%d %d", &a, &b);
    printf("\n\n\t-----------------------------");
    printf("\n\n\t press 1 for addition");
    printf("\n\n\t press 2 for subtraction");
    printf("\n\n\t press 3 for multiply");
    printf("\n\n\t press 4 for division");
    printf("\n\n\t enter your choice :");
    scanf("%d", &choice);
```

```
    printf("\n\n\t------------------------------");

    switch (choice)
    {
    case 1: printf("\n\n\t addition :%d", a + b);
        break;
     case 2: printf("\n\n\t subtraction :%d", a - b);
        break;
    case 3: printf("\n\n\t multiply :%d", a * b);
        break;
    case 4: printf("\n\n\t division :%d", a / b);
        break;
    default: printf("\n\n\t invalid choice ");
    }
}
```

LAB EXERCISE 2: Check Number Properties

• Write a C program that takes an integer from the user and checks the following using different operators:

Ans.

o Whether the number is even or odd.

```
#include <stdio.h>

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);
  }
```

o Whether the number is a multiple of both 3 and 5.

```
#include<stdio.h>
main()
{
        int a;
        printf("\n\n\t enter a number :");
        scanf("%d", &a);
        if (a % 3 == 0 && a % 5 == 0)
                printf("\n\n\t number %d is div by both 3 and 5",a);
        else if(a%3==0)
                printf("\n\n\t number %d is div by  3 only",a);
        else if(a%5==0)
                printf("\n\n\t number %d is div by  5 only ",a);
        else
                printf("\n\n\t number %d is not div by both 3 and 5",a);
}
```

o Whether the number is positive, negative, or zero.

```
#include<stdio.h>
main()
{
        int number;
        printf("\n\n\t enter a number :");
        scanf("%d", &number);
```

```
    if (number > 0)
        printf("\n\n\t your number is positive :%d", number);
    else if (number == 0)
        printf("\n\n\t your number is zero :%d", number);
    else
        printf("\n\n\t your number is negative :%d", number);
}
```

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.

Ans.

```
#include<stdio.h>
main()
{
    int id, m1, m2, m3, m4, m5 ,total ;
    float per ;
    char str[30];
    printf("\n\n\t enter your Id :");
    scanf("%d" ,& id);
    printf("\n\n\t enter your Name :");
```

```
scanf("%s" , & str);
printf("\n\n\t enter marks of English :");
scanf("%d" , & m1);
printf("\n\n\t enter marks of Maths :");
scanf("%d", &m2);
printf("\n\n\t enter marks of Science :");
scanf("%d", &m3);
printf("\n\n\t enter marks of History :");
scanf("%d", &m4);
printf("\n\n\t enter marks of Hindi :");
scanf("%d", &m5);

total = m1 + m2 + m3 + m4 + m5;
per = total / 5;
printf("\n\n\t ------------------------------");
printf("\n\n\t ---------MARKS SHEET----------");
printf("\n\n\t ------------------------------");
printf("\n\n\t your Id            :%d", id);
printf("\n\n\t your Name          :%s", str);
printf("\n\n\t ------------------------------");
printf("\n\n\t Marks in English    :%d", m1);
printf("\n\n\t Marks in Maths      :%d", m2);
printf("\n\n\t Marks in Science    :%d", m3);
printf("\n\n\t Marks in History    :%d", m4);
printf("\n\n\t Marks in Hindi      :%d", m5);
```

```
        printf("\n\n\t ----------------------------");
        printf("\n\n\t Total Marks        :%d", total);
        printf("\n\n\t Percentage         :%.2f", per);

        if (per >= 70)
                printf("\n\n\t your grade is A+");
        else if (per >= 60)
                printf("\n\n\t your grade is A");
        else if (per >= 50)
                printf("\n\n\t your grade is B");
        else if (per >= 40)
                printf("\n\n\t your grade is C");
        else
                printf("\n\n\t fail");
        printf("\n\n\t ----------------------------");
}
```

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.

Ans.

```
#include<stdio.h>
main()
{
```

```
	int a, b, c;
	printf("\n\n\t enter first number :");
	scanf("%d", &a);
	printf("\n\n\t enter second number :");
	scanf("%d", &b);
	printf("\n\n\t enter third number :");
	scanf("%d", &c);
	if (a > b)
	{
		if (a > c)
			printf("\n\n\t first number is maximum :%d", a);
		else
			printf("\n\n\t first number is not max :%d", a);
	}
	else if (b > c)
	{
		if (b > a)
			printf("\n\n\t second number is maximum :%d", b);
		else
			printf("\n\n\t second number is not max :%d", b);
	}
	else
		printf("\n\n\t third number is maximum :%d", c);
}
```

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.**

Ans.

```
#include <stdio.h>
 main() {
   int num, i, status = 0;
   printf("Enter a number: ");
   scanf("%d", &num);
   if (num < 2) {
      printf("%d is not a prime number.\n", num);
   }
 else {
      for (i = 2; i <= num - 1; i++) {
         if (num % i == 0) {
            status = 1;
            break;
         }
      }
      if (status == 1)
     {
```

```
        printf("%d is not a prime number.\n", num);
    }
 else
   {
        printf("%d is a prime number.\n", num);
    }
  }
}
```

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).

Ans.

```
#include <stdio.h>
 main() {
  int num;
  printf("Enter a number: ");
  scanf("%d", &num);
  printf("Multiplication table of : %d", num);
  for (int i = 1; i <= 10; i++) {
     printf("%d x %d = %d ", num, i, num * i);
  }
}
```

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.**

Ans.

```
#include <stdio.h>


int main() {
   int num, sum = 0,org,rem=0;
   printf("Enter a number: ");
   scanf("%d", &num);
   org = num;
   while (num != 0) {
      sum += num % 10;
      num /= 10;
   }
   printf("Sum of digits: %d\n", sum);
   num = org;
   while (num > 0)
   {
      rem = num % 10;
      printf("%d", rem);
      num = num / 10;
   }
}
```

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.

Ans.

```
#include<stdio.h>
main()
{
	int arr[30], size, i,num,status=0,high=0;
	printf("\n\n\t enter size of array :");
	scanf("%d", &size);
	for (i = 0;i < size;i++)
	{
		printf("\n\n\t enter array element arr[%d] :", i);
		scanf("%d", &arr[i]);
	}
	for (i = 0;i < size;i++)
	{
		printf("\n\n\t your array element arr[%d] is : %d", i, arr[i]);
	}
	for (i = 0;i < size;i++)
	{
		if (arr[i] > high)
		{
```

```
                high = arr[i];
            }
        }
        printf("\n\n\t enter element you want to check for maximum :");
        scanf("%d", &num);
        if (num == high)
            printf("\n\n\t entered  number is maximum ");
        else
            printf("\n\n\t entered number is minimum ");
}
```

LAB EXERCISE 2: 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.

Ans.

```
#include<stdio.h>
main()
{
    int arr[30], size, i , sum=0;

    printf("\n\n\t enter size of array :");
    scanf("%d", &size);

    for (i = 0;i < size;i++)
    {
```

```
            printf("\n\t enter elements [%d] :", i);
            scanf("%d", &arr[i]);
        }
        for (i = 0;i < size;i++)
        {
            printf("\n\t your elements in array [%d] : %d", i, arr[i]);
            sum = sum + arr[i];
            printf("\n\t sum of  array elements : %d", sum);
        }
}
```

6. Strings

LAB EXERCISE 1: String Reversal

• Write a C program that takes a string as input and reverses it using a function.

Ans.

```
#include <stdio.h>
#include <string.h>
void reverse_string(char str[]) {
   int length = strlen(str);
   int start = 0;
   int end = length - 1;
   while (start < end)
   {
      char temp = str[start];
```

```
        str[start] = str[end];
        str[end] = temp;
        start++;
        end--;
    }
}
int main() {
    char str[100];
    printf("Enter a string: ");
    scanf("%[^\n]s", str);
    printf("Original string: %s\n", str);
    reverse_string(str);
    printf("Reversed string: %s\n", str);
}
```

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.

Ans.

```
#include<stdio.h>
main()
{
        char ch;
        printf("\n\n\t enter a character  :");
        scanf("%c", &ch);
```

```
        switch (ch)
        {
        case 'a':
                printf("\n\n\t it is a vowel");
                break;
        case 'e':
                printf("\n\n\t it is a vowel");
                break;

        case 'i':
                printf("\n\n\t it is a vowel");
                break;
        case 'o': if (ch == 'o' || ch == 'O')
                printf("\n\n\t it is a vowel");
                break;
        case 'u': if (ch == 'u' || ch == 'u')
                printf("\n\n\t it is a vowel");
                break;
        default:printf("\n\n\t it is consonants");
                break;
        }
}
```

LAB EXERCISE 3:

Word Count • Write a C program that counts the number of words in a sentence entered by the user.

Ans.

```
#include <stdio.h>
main() {
   char sen[100];
   int word_count = 0;
   printf("Enter a sentence: ");
   scanf("%[^\n]s", sen);

   int i = 0;
   while (sen[i] != '\0') {
      if (sen[i] == ' ' && sen[i + 1] != ' ' && sen[i + 1] != '\0') {
         word_count++;
      }
      i++;
   }
   if (sen[0] != ' ') {
      word_count++;
   }
   printf("Number of words: %d\n", word_count);
}
```