Module 2 – Introduction to Programming

1) Overview of C Programming

THEORY EXERCISE:

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

Answer :-

C programming, created by Dennis Ritchie in the 1970s at Bell Labs, is one of the most important programming languages in history. C is still important for its speed, portability, and ability to handle low-level tasks in systems and applications.

LAB EXERCISE:

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

Answer :-

Real-World Applications of C Programming

A) Embedded Systems

- C is widely used in embedded systems, such as microcontrollers in automotive systems, home appliances, and medical devices. Its efficiency and ability to interact with hardware make it ideal for programming devices with limited resources.

B) Operating Systems

- Many operating systems, including UNIX, Linux, and portions of Windows, are written in C. Its low-level capabilities allow direct access to hardware while maintaining portability across different platforms.

C) Game Development

- C is used in game engines like Unity (alongside C++) for high-performance game development. Its speed and control over system resources ensure smooth rendering and real-time responsiveness, critical for gaming applications.

2) Setting Up Environment

THEORY EXERCISE:

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

Answer :-

Step 1: Install a C Compiler (e.g., GCC)

Download GCC:

- Go to the GCC official website or install it via package managers like MinGW on Windows, or apt on Linux (sudo apt install gcc).

Install the Compiler:

- Follow the installation steps for your operating system, ensuring it is added to your system's PATH.

Step 2: Choose and Install an IDE

Download the IDE:

- Select an IDE like DevC++, VS Code, or CodeBlocks. Visit their official websites for the installer.

Install the IDE:

- Run the installer and follow the on-screen instructions.

Step 3: Configure the IDE

Link the Compiler:

- Open the IDE and navigate to its settings/preferences. Set Compiler Path: Point the IDE to the installed GCC compiler's location.

Create a Project:

- Start a new C project and test the configuration.

LAB EXERCISE:

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

Answer :-

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

3. Basic Structure of a C Program

THEORY EXERCISE:

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

Answer :-

A) document section

- The comment of the documentation section is written in comment line and at the time of excuation of program it will be ingnored.

- <u>There Two types write comment</u>

1) single line Comment Line

- // Hello

2) Multiple Line Comment Line

- /* hello

C Language */

B) Link Section

- The link section provides instrcution to the compiler to link function from the system library.

- The header file needs to add in our program to use Predifined functio of C.

EX:-

#include<stdio.h> { stdio.h means Standard Input output Header file.

C) Definition Section

- The Defination Section defines all Symbolic Constants.

- we may often use certain unique constants in a program .

EX:-

#define PI 3.1416

#define MAX 180

D) Main () Function

- Every C Program must have one main() function.

- Main Function contains two Parts:-

1) declaration

2) Executable

-The Declartion part declares all the varibles used in the excutable part.

-There is least one statement in the excutable part.

-these two parts must apper between the opening and closing braces.

E) Data Types:

- C has a concept of 'data types' which are used to defined a varible before its use.

- The value of variable can be chnage any time.

- ANSI C supports three classes of Data Types:

A) Primary Data Type

B) Derived Data Type

3) User-define Data Type

LAB EXERCISE:

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

Answer :-

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

```
int main() {
  int age = 25;
float height = 5.9;
char grade = 'A';
const float PI = 3.14159;
printf("Age: %d\n", age);
printf("Height: %.1f\n", height);
printf("Grade: %c\n", grade);
printf("Value of PI: %.5f\n", PI);
return 0;
}
```

4. Operators in C

THEORY EXERCISE:

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

ANswer :-

A) Airthmetic Operators

- C Provides all the basic arithmetic operators.

- These can operate on any built-in data types allowed in C.

- These Operators +,-,*,/ All work the same way as they do in other language.

B) Relation Operator

- To Compare two values and depending on their relation, take certain decision.

C) Logical

- The logical Operators && and || are used when we want to test more than one condition and make Decision.

- Ex a> b && x==10

D) Assingment

- Assingment operator used to assing the result of an expresssion to a varible.

-EX: a=10

-we have seen assingment operator'='.

E) Increment and Decrement Opeator

- These are increent and decrement operators:

++ and --

-The operator ++ adds to the operand , while --subtracks 1.

-Both are unary operators and takes the follwing form.

EX: ++m; or m++;

--m; or m--;

- ++m; is equivalent to m=m+1

- --m; is equivalent to m=m-1

-A Prefix operator first add 1 to the operand and then the result is assing to the varible on left.

Ex: ++m; or --m

-A postfix operator first assings the value to the variable on left and then increment the operand

EX: m++ or m--

F) Conditional

- Ternary operator for short-hand conditional expressions.

- EX :

Syntax: condition ? expression1 : expression2.

LAB EXERCISE:

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

Answer :

```
#include <stdio.h>

int main() {

int num1, num2;

printf("Enter two integers: ");
```

```
    scanf("%d %d", &num1, &num2);

    printf("\nArithmetic Operations:\n");
    printf("Addition: %d + %d = %d\n", num1, num2, num1 + num2);
    printf("Subtraction: %d - %d = %d\n", num1, num2, num1 - num2);
    printf("Multiplication: %d * %d = %d\n", num1, num2, num1 * num2);
    printf("Division: %d / %d = %d\n", num1, num2, num1 / num2);

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

    printf("\nLogical Operations:\n");
    printf("(%d > 0) && (%d > 0) = %d\n", num1, num2, (num1 > 0) && (num2
> 0));
    printf("(%d < 0) || (%d > 0) = %d\n", num1, num2, (num1 < 0) || (num2 >
0));
    return 0;
}
```

5. Control Flow Statements in C

THEORY EXERCISE:

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

Answer:

A) if statement

- The if statment is a powerful decision making statment and is used to control the flow of execution of statments.

- it is basically a two-way decision statment and is used in conjunction with an expression.

EX :

```
int num = 5;
if (num > 0) {
   printf("Positive number\n");
}
```

B) If else statment

-if the test expression is true, then the true-block statemnt , immedistely following the if statement are excuted;

otherwise the false block statment are executed.

-EX

```
int num = 1;
if (num > 0) {
   printf("Positive number\n");
} else {
```

```
    printf("Negative number\n");
  }
```

C) Nested if-else

- Allows multiple levels of decision-making.
- EX

```
int num = 0;
if (num > 0) {
   printf("Positive number\n");
 } else if (num == 0) {
   printf("Zero\n");
 } else {
   printf("Negative number\n");
 }
```

LAB EXERCISE:

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.).

ANswer :

```
#include <stdio.h>
int 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("\nEnter 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\n");
 }
```

```
return 0;
}
```

6. Looping in C

THEORY EXERCISE:

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

Answer :-

1.While Loop

- Used when the number of iterations is unknown, and the condition must be checked before executing the loop body.

-EX:

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

2.For Loop

- Ideal when the number of iterations is known beforehand.

-EX:

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

3.Do-While Loop

-Ensures the loop body executes at least once, regardless of the condition.

-EX:

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

LAB EXERCISE:

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

Answer :

```
#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

THEORY EXERCISE:

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

Answer:-

1.Breack Statment

- Used to exit a loop or switch statement prematurely.

-EX:

```
for (int i = 1; i <= 10; i++) {
if (i == 5) {
break;
}
printf("%d ", i);
```

```
}
```

2.Continue Statement

- Skips the remaining code in the current iteration and proceeds to the next iteration.

-EX:

```
for (int i = 1; i <= 10; i++) {
if (i == 3) {
continue;
}
printf("%d ", i);
}
```

3.Goto Statement

- Transfers control to a labeled statement elsewhere in the code. While it can simplify some scenarios, it is generally discouraged for readability and maintainability.

-EX:

```
int i = 1;
start:
printf("%d ", i);
i++;
if (i <= 5) {
goto start;
}
```

LAB EXERCISE:

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.

Answer:-

```
#include <stdio.h>

int main() {

printf("Using break statement:\n");

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

if (i == 5) {

break;

}

printf("%d ", i);

}

printf("\n");

printf("Using continue statement:\n");

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

if (i == 3) {

continue;

}

printf("%d ", i);

}

printf("\n");

return 0;

}
```

8. Functions in C

THEORY EXERCISE:

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

Answer:-

- Functions in C are reusable blocks of code that perform specific tasks. They help in organizing the program, improving readability, and reducing code duplication.

- A) Function Declaration:

  - Specifies the function name, return type, and parameters.

  -EX:

```
int add(int a, int b);  // Declaration
```

B) Function Definition:

  - Contains the actual implementation of the function.

  -EX:

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

C) Function Call:

  - Executes the function by passing required arguments.

  -EX:

```
int result = add(5, 10);  // Call
printf("Sum: %d\n", result);
```

LAB EXERCISE:

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

Answer:-

```
#include <stdio.h>

int factorial(int n);  // Function Declaration

int main() {

int num;

printf("Enter a positive integer: ");

scanf("%d", &num);

printf("Factorial of %d is %d\n", num, factorial(num));  // Function Call

return 0;

}

int factorial(int n) {   // Function Definition

if (n == 0 || n == 1) {

return 1;  // Base case: Factorial of 0 or 1 is 1

}

return n * factorial(n - 1);  // Recursive call

}
```

9. Arrays in C

THEORY EXERCISE:

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

Answer:-

- An array is a collection of elements of the same data type stored in contiguous memory locations. Arrays allow storing multiple values under a single variable name.

- A) One-Dimensional Array

- Stores data in a linear structure.

- A list items or data can be given one varible name using only one subscript ans such a variable called a One- Dimensional Array

- Declaration:

```
data_type array_name[size];
```

-EX:

```
int numbers[5] = {1, 2, 3, 4, 5};
for (int i = 0; i < 5; i++) {
printf("%d ", numbers[i]);
```

- B) Multi-Dimensional Array

- Stores data in a tabular form (rows and columns).

- C allows array of more than two imensional, these are known as multi dimensional array.

- Declaration:

```
data_type array_name[rows][columns];
```

-EX:

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

```
for (int i = 0; i < 2; i++) {
 for (int j = 0; j < 2; j++) {
 printf("%d ", matrix[i][j]);
 }
 }
```

LAB EXERCISE:

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.

Answer :

```
#include <stdio.h>
int main() {
// One-dimensional array
int arr[5], i;
printf("Enter 5 integers:\n");
for (i = 0; i < 5; i++) {
scanf("%d", &arr[i]);
}
printf("Elements of the one-dimensional array:\n");
for (i = 0; i < 5; i++) {
printf("%d ", arr[i]);
}
printf("\n");
// Two-dimensional array (3x3 matrix)
int matrix[3][3], sum = 0;
```

```
printf("Enter elements of a 3x3 matrix:\n");
for (i = 0; i < 3; i++) {
for (int j = 0; j < 3; j++) {
scanf("%d", &matrix[i][j]);
sum += matrix[i][j];
}
}
printf("Elements of the 3x3 matrix:\n");
for (i = 0; i < 3; i++) {
for (int j = 0; j < 3; j++) {
printf("%d ", matrix[i][j]);
}
printf("\n");
}
printf("Sum of all elements in the matrix: %d\n", sum);
return 0;
}
```

10. Pointers in C

THEORY EXERCISE:

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

Answer:

- Pointers in C are variables that store the memory address of another variable.
- They allow direct memory manipulation and are a powerful feature of C.
- Declaration:

data_type *pointer_name;

- Initialization:

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

- EX:

```
int a = 10;
int *p = &a;
```

- Importance of Pointers:

A) Memory Efficiency

- Access and modify variables directly through memory addresses.

B) Dynamic Memory Allocation

- Allocate and manage memory during runtime using malloc and free.

C) Function Arguments

- Pass variables by reference to functions, avoiding duplication.

D) Data Structures

- Enable implementation of advanced structures like linked lists, trees, etc.

LAB EXERCISE:

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

Answer :

```
#include <stdio.h>
int main() {
int num = 10;
int *ptr = &num;
printf("Original value of num: %d\n", num);
printf("Address of num: %p\n", (void *)&num);
```

```
*ptr = 20;
printf("Modified value of num using pointer: %d\n", num);
printf("Pointer's value (address it points to): %p\n", (void *)ptr);
printf("Value at the address stored in pointer: %d\n", *ptr);
return 0;
}
```

11. Strings in C

THEORY EXERCISE:

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

Answer:

- Strings in C are arrays of characters terminated by a null character (\0). The standard library <string.h> provides various functions for string manipulation.

- A) strlen()

  - Returns the length of a string (excluding the null terminator).

  - EX:

```
char str[] = "Hello";
printf("Length: %ld\n", strlen(str));
// Output: 5
```

B) strcpy()

  - Copies the content of one string to another.

  - EX:

```
char source[] = "World";
char destination[20];
strcpy(destination, source);
```

```
printf("Copied String: %s\n", destination);
// Output: World
```

C) strcat()

- Concatenates (appends) one string to the end of another.
- EX:

```
char str1[20] = "Hello ";
char str2[] = "World";
strcat(str1, str2);
printf("Concatenated String: %s\n", str1);
// Output: Hello World
```

D) strcmp()

- Compares two strings lexicographically. Returns 0 if equal, a positive value if the first string is greater, or a negative value otherwise.
- EX:

```
if (strcmp("apple", "banana") < 0) {
printf("apple comes before banana\n");
}
```

E) strchr()

- Finds the first occurrence of a character in a string.
- EX:

```
char str[] = "Hello World";
char *pos = strchr(str, 'o');
printf("First 'o' found at: %ld\n", pos - str);
```

```
// Output: 4
```

LAB EXERCISE:

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().

Answer :

```
#include <stdio.h>

#include <string.h>

int main() {

char str1[50], str2[50];

// Input two strings from the user

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';

// Concatenate strings using strcat()

strcat(str1, str2);

// Display concatenated string and its length

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

printf("Length of concatenated string: %ld\n", strlen(str1));

return 0;

}
```

12. Structures in C

THEORY EXERCISE:

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

Answer:

- A structure in C is a user-defined data type that groups different types of variables under one name. It is useful for representing complex data in an organized way.

- Declaring a Structure:

  - Use the struct keyword to define a structure.

  - struct structure_name {

    data_type member1;

    data_type member2;

    };

- Initializing a Structure:

  - Initialize members when defining a structure variable.

  - EX:

    struct Student {

    char name[50];

    int roll_number;

    float marks;

    };

- Accessing Structure Members:

  - Use the dot operator (.) with the structure variable.

  - EX:

```
printf("Name: %s\n", s1.name);
printf("Roll Number: %d\n", s1.roll_number);
printf("Marks: %.2f\n", s1.marks);
```

```
struct Student s1 = {"Alice", 101, 85.5};
```

LAB EXERCISE:

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.

Answer :-

```
#include <stdio.h>
struct Student {
char name[50];
int roll_number;
float marks;
};
int main() {
struct Student students[3];  // Array of 3 structures
// Input details of 3 students
for (int i = 0; i < 3; i++) {
printf("Enter details for student %d:\n", i + 1);
printf("Name: ");
scanf(" %[^\n]", students[i].name);  // Read string with spaces
printf("Roll Number: ");
scanf("%d", &students[i].roll_number);
printf("Marks: ");
```

```
scanf("%f", &students[i].marks);
}
printf("\nDetails of students:\n");
for (int i = 0; i < 3; i++) {
printf("Student %d:\n", i + 1);
printf("Name: %s\n", students[i].name);
printf("Roll Number: %d\n", students[i].roll_number);
printf("Marks: %.2f\n\n", students[i].marks);
}
return 0;
}
```

13. File Handling in C

THEORY EXERCISE:

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

Answer :-

- File handling in C is essential for storing, retrieving, and processing data in files. It allows data to persist beyond program execution, enabling tasks like data logging, configuration management, and record-keeping.

- File Operations:

A) Opening a File:

- Use fopen() to open a file in different modes:

"r": Read mode (file must exist).

"w": Write mode (creates a new file or overwrites an existing file).

"a": Append mode (writes at the end of the file).

"r+", "w+", "a+": Read and write modes.

- EX:

FILE *file = fopen("example.txt", "w");

B) Writing to a File:

- Use fprintf(), fputs(), or fwrite() to write data.
- EX:

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

C) Reading from a File:

- Use fscanf(), fgets(), or fread() to read data.
- char buffer[100];

fgets(buffer, sizeof(buffer), file);

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

D) Closing a File:

- Use fclose() to release resources after file operations.
- Ex

fclose(file);

LAB EXERCISE:

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.

Answer :-

```
#include <stdio.h>
int main() {
FILE *file;
```

```
char content[] = "This is a sample file content.";

char buffer[100];

file = fopen("sample.txt", "w");

if (file == NULL) {

printf("Error: Could not open file for writing.\n");

return 1;

}

fprintf(file, "%s", content);  // Writing to file

fclose(file);  // Closing the file

file = fopen("sample.txt", "r");

if (file == NULL) {

printf("Error: Could not open file for reading.\n");

return 1;

}

fgets(buffer, sizeof(buffer), file);  // Reading from file

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

fclose(file);  // Closing the file

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.

Answer :

```
#include <stdio.h>

int main() {

float num1, num2, result;

char operator;

printf("Enter the first number: ");

scanf("%f", &num1);

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

scanf(" %c", &operator);  // Space before %c to handle newline character

printf("Enter the second number: ");

scanf("%f", &num2);

switch (operator) {

  case '+':

      result = num1 + num2;

      printf("Result: %.2f\n", result);
```

```
        break;
    case '-':
        result = num1 - num2;
        printf("Result: %.2f\n", result);
        break;
    case '*':
        result = num1 * num2;
        printf("Result: %.2f\n", result);
        break;
    case '/':
        if (num2 != 0) {
        result = num1 / num2;
        printf("Result: %.2f\n", result);
        } 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: Division by zero is not allowed.\n");
        }
        break;
        default:
```

```
printf("Error: Invalid operator.\n");
}
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, orzero.
- o Whether the number is a multiple of both 3 and 5.

ANswer :-

```
#include <stdio.h>
int main() {
int num;
printf("Enter an integer: ");
scanf("%d", &num);
if (num % 2 == 0) {
printf("The number is even.\n");
} else {
printf("The number is odd.\n");
}
if (num > 0) {
printf("The number is positive.\n");
```

```
} else if (num < 0) {
printf("The number is negative.\n");
} else {
printf("The number is zero.\n");
}
if (num % 3 == 0 && num % 5 == 0) {
printf("The number is a multiple of both 3 and 5.\n");
} else {
printf("The number is not a multiple of both 3 and 5.\n");
}
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.

ANSWER :-

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

```
printf("Enter the marks of the student: ");
scanf("%d", &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:

- The largest number.
- The smallest number.

Challenge: Solve the problem using both if-else and switch-case statements.

Answer :-

```
#include <stdio.h>
int main() {
int num1, num2, num3;
printf("Enter three numbers: ");
scanf("%d %d %d", &num1, &num2, &num3);
```

```
    if (num1 >= num2 && num1 >= num3) {
        printf("The largest number is: %d\n", num1);
    } else if (num2 >= num1 && num2 >= num3) {
        printf("The largest number is: %d\n", num2);
    } else {
        printf("The largest number is: %d\n", num3);
    }

    if (num1 <= num2 && num1 <= num3) {
        printf("The smallest number is: %d\n", num1);
    } else if (num2 <= num1 && num2 <= num3) {
        printf("The smallest number is: %d\n", num2);
    } else {
        printf("The smallest number is: %d\n", num3);
    }
     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.

ANSWER :-

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

```
int main() {
int num, i, j, isPrime;
  printf("Enter a number to check if it's prime: ");
  scanf("%d", &num);
if (num <= 1) {
   printf("%d is not a prime number.\n", num);
} else {
isPrime = 1;
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("Enter the upper limit to print all primes between 1 and that number: ");
scanf("%d", &num);
printf("Prime numbers between 1 and %d are:\n", num);
for (i = 2; i <= num; i++) {
 isPrime = 1;  // Assume i is prime
```

```
    for (j = 2; j <= i / 2; j++) {
     if (i % j == 0) {
     isPrime = 0; // Not a prime number
     break;
    }
   }
    if (isPrime) {
       printf("%d ", i);
     }
   }
  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).

Answer :-

```
#include <stdio.h>
int main() {
int num, i, N;
 printf("Enter the number for multiplication table: ");
 scanf("%d", &num);
 printf("Enter the range (N) for the multiplication table (e.g., 1 to N): ");
 scanf("%d", &N);
```

```
 printf("Multiplication table of %d from 1 to %d is:\n", num, N);
for (i = 1; i <= N; 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

ANSWER :-

```
#include <stdio.h>
int main() {
int num, sum = 0, reverse = 0, digit;
printf("Enter an integer: ");
scanf("%d", &num);
int originalNum = num;

 while (num != 0) {
   digit = num % 10;
   sum += digit;
   reverse = reverse * 10 + digit;
   num /= 10;
}
```

```
printf("Sum of the digits of %d: %d\n", originalNum, sum);
printf("Reversed number of %d: %d\n", originalNum, 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.

ANSWER:-

```
#include <stdio.h>
int main() {
int arr[10], i, j, temp, max, min;
 printf("Enter 10 integers:\n");
for (i = 0; i < 10; i++) {
 scanf("%d", &arr[i]);
}
 max = arr[0];
 min = arr[0];
 for (i = 1; i < 10; i++) {
  if (arr[i] > max) {
    max = arr[i];
  }
```

```
    if (arr[i] < min) {
    min = arr[i];
   }
  }
 printf("Maximum value: %d\n", max);
 printf("Minimum value: %d\n", min);

for (i = 0; i < 9; i++) {
 for (j = i + 1; j < 10; j++) {
  if (arr[i] > arr[j]) {
    temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
  }
 }
}

 printf("Sorted array in ascending order:\n");
 for (i = 0; i < 10; 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.

ANSWER:-

```
#include <stdio.h>
 int main() {
 int matrix1[3][3], matrix2[3][3], sum[3][3], product[3][3];
 int i, j, k;

 printf("Enter elements of the first 3x3 matrix:\n");
  for (i = 0; i < 3; i++) {
   for (j = 0; j < 3; j++) {
     scanf("%d", &matrix1[i][j]);
    }
    }

printf("Enter elements of the second 3x3 matrix:\n");
for (i = 0; i < 3; i++) {
 for (j = 0; j < 3; j++) {
  scanf("%d", &matrix2[i][j]);
  }
```

```
      }


    for (i = 0; i < 3; i++) {
     for (j = 0; j < 3; j++) {
       sum[i][j] = matrix1[i][j] + matrix2[i][j];
     }
    }


     for (i = 0; i < 3; i++) {
        for (j = 0; j < 3; j++) {
           product[i][j] = 0;
              for (k = 0; k < 3; k++) {
            product[i][j] += matrix1[i][k] * matrix2[k][j];
          }
         }
        }

printf("Sum of the two matrices:\n");
for (i = 0; i < 3; i++) {
   for (j = 0; j < 3; j++) {
      printf("%d ", sum[i][j]);
   }
   printf("\n");
}
```

```
    printf("Product of the two matrices:\n");
    for (i = 0; i < 3; i++) {
        for (j = 0; j < 3; j++) {
            printf("%d ", product[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

ANSWER:-

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

```
for (i = 0; i < n; i++) {
 scanf("%d", &arr[i]);
}
for (i = 0; i < n; i++) {
 sum += arr[i];
}
 average = sum / n;

printf("Sum of the array elements: %.2f\n", sum);
printf("Average of the array 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.

Answer :-

```
#include <stdio.h>
int fibonacci_recursive(int n) {
if (n <= 1)
  return n;
  return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2);
```

```
    }

    int fibonacci_iterative(int n) {
     int a = 0, b = 1, next, i;
     if (n == 0) return a;
      for (i = 2; i <= n; i++) {
      next = a + b;
     a = b;
     b = next;
      }
     return b;
    }

int main() {
int n, i;
printf("Enter the number of terms (N): ");
scanf("%d", &n);
printf("Fibonacci sequence up to %d terms (Recursive):\n", n);
for (i = 0; i < n; i++) {
   printf("%d ", fibonacci_recursive(i));
}
printf("\n");
printf("Nth Fibonacci number (Recursive): %d\n", fibonacci_recursive(n - 1));
printf("Nth Fibonacci number (Iterative): %d\n", fibonacci_iterative(n - 1));
```

```
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.

Answer :-

```
#include <stdio.h>
unsigned long long factorial_recursive(int n) {
if (n == 0 || n == 1)
return 1;
return n * factorial_recursive(n - 1);
}

unsigned long long factorial_iterative(int n) {
unsigned long long fact = 1;
for (int i = 1; i <= n; i++) {
fact *= i;
}
return fact;
}

int main() {
int num;
```

```
    printf("Enter a positive integer: ");
    scanf("%d", &num);

    if (num < 0) {
        printf("Factorial is not defined for negative numbers.\n");
    } else {
        unsigned long long fact_recursive = factorial_recursive(num);
        unsigned long long fact_iterative = factorial_iterative(num);
        printf("Factorial of %d (Recursive): %llu\n", num, fact_recursive);
        printf("Factorial of %d (Iterative): %llu\n", num, fact_iterative);
    }
    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.

Answer :

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
bool isPalindromeNumber(int num) {
 int original = num, reversed = 0, remainder;
  while (num != 0) {
 remainder = num % 10;
```

```
        reversed = reversed * 10 + remainder;
        num /= 10;
      }
     return original == reversed;
    }
     bool isPalindromeString(char str[]) {
     int len = strlen(str);
     for (int i = 0; i < len / 2; i++) {
      if (str[i] != str[len - 1 - i]) {
       return false;
      }
     }
     return true;
    }
int main() {
    int choice;
    printf("Choose an option:\n");
    printf("1. Check if a number is a palindrome\n");
    printf("2. Check if a string is a palindrome\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    if (choice == 1) {
        int num;
        printf("Enter a number: ");
```

```
        scanf("%d", &num);
        if (isPalindromeNumber(num)) {
            printf("The number %d is a palindrome.\n", num);
        } else {
            printf("The number %d is not a palindrome.\n", num);
        }
    } else if (choice == 2) {
        char str[100];
        printf("Enter a string: ");
        scanf("%s", str);
        if (isPalindromeString(str)) {
            printf("The string \"%s\" is a palindrome.\n", str);
        } else {
            printf("The string \"%s\" is not a palindrome.\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.

Answer :

```
#include <stdio.h>
void reverseString(char str[]) {
    int length = 0, i;
    char temp;
    while (str[length] != '\0') {
        length++;
    }
    for (i = 0; i < length / 2; i++) {
        temp = str[i];
        str[i] = str[length - 1 - i];
        str[length - 1 - i] = temp;
    }
}
int main() {
    char str[100];
    printf("Enter a string: ");
    scanf("%s", 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.

ANSWER:-

```
#include <stdio.h>

#include <ctype.h>

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

    *vowels = *consonants = *digits = *special = 0;

    for (int i = 0; str[i] != '\0'; i++) {

        char ch = tolower(str[i]);  // Convert to lowercase for easier comparison

        if ((ch >= 'a' && ch <= 'z')) {

            // Check for vowels

            if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {

                (*vowels)++;

            } else {

                (*consonants)++;

            }

        } else if (ch >= '0' && ch <= '9') {

            (*digits)++;

        } else {

            (*special)++;

        }

    }

}
```

```
int main() {
    char str[100];
    int vowels, consonants, digits, special;
    printf("Enter a string: ");
    scanf("%[^\n]", str);  // Read until newline to include spaces in input
    countCharacters(str, &vowels, &consonants, &digits, &special);
    printf("Vowels: %d\n", vowels);
    printf("Consonants: %d\n", consonants);
    printf("Digits: %d\n", digits);
    printf("Special characters: %d\n", special);
    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.

ANSWER :-

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

int main() {
    char sentence[1000], word[100], longestWord[100] = "";
    int wordCount = 0, maxLength = 0, length = 0, index = 0;
    printf("Enter a sentence: ");
    fgets(sentence, sizeof(sentence), stdin);
    for (int i = 0; sentence[i] != '\0'; i++) {
```

```
        if (sentence[i] != ' ' && sentence[i] != '\n') {
            word[index++] = sentence[i]; // Build the current word
        } else {
            if (index > 0) {
                word[index] = '\0';
                wordCount++;
                length = strlen(word);
                if (length > maxLength) {
                    maxLength = length;
                    strcpy(longestWord, word);
                }
                index = 0;
            }
        }
    }
    printf("The sentence has %d word(s).\n", wordCount);
    printf("The longest word is '%s' with a length of %d.\n", longestWord, maxLength);
    return 0;
}
```

Extra Logic Building Challenges

Lab Challenge 1: Armstrong Number

Write a C program that checks whether a given number is an Armstrong number or not (e.g.,153 = 1^3 + 5^3 + 3^3).

Challenge: Write a program to find all Armstrong numbers between 1 and 1000.

ANSWER :-

```
#include <stdio.h>

#include <math.h>

int isArmstrong(int num) {

   int originalNum = num, sum = 0, digit;

   int numDigits = 0;

   while (originalNum > 0) {

      numDigits++;

      originalNum /= 10;

   }

   originalNum = num;

   while (originalNum > 0) {

      digit = originalNum % 10;

      sum += pow(digit, numDigits);

      originalNum /= 10;

   }

   return (sum == num);

}

int main() {

   int num;
```

```
    printf("Enter a number to check if it is an Armstrong number: ");
    scanf("%d", &num);

    if (isArmstrong(num)) {
        printf("%d is an Armstrong number.\n", num);
    } else {
        printf("%d is not an Armstrong number.\n", num);
    }
    printf("Armstrong numbers between 1 and 1000 are:\n");
    for (int i = 1; i <= 1000; i++) {
        if (isArmstrong(i)) {
            printf("%d ", i);
        }
    }
    printf("\n");
    return 0;
}
```

Lab Challenge 2: Pascal's Triangle

Write a C program that generates Pascal's Triangle up to N rows using loops.

Challenge: Implement the same program using a recursive function.

ANSWER :-

```
#include <stdio.h>
int factorial(int n) {
    if (n == 0 || n == 1)
```

```
        return 1;
    return n * factorial(n - 1);
}
void generatePascalLoop(int n) {
    printf("Pascal's Triangle (Using Loops):\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j <= i; j++) {
            printf("%d ", factorial(i) / (factorial(j) * factorial(i - j)));
        }
        printf("\n");
    }
}
int binomialCoeff(int n, int k) {
    if (k == 0 || k == n)
        return 1;
    return binomialCoeff(n - 1, k - 1) + binomialCoeff(n - 1, k);
}
void generatePascalRecursive(int n) {
    printf("Pascal's Triangle (Using Recursion):\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j <= i; j++) {
            printf("%d ", binomialCoeff(i, j));
        }
        printf("\n");
    }
```

```
}


int main() {
    int n, choice;
    printf("Enter the number of rows for Pascal's Triangle: ");
    scanf("%d", &n);
    printf("Choose the method:\n1. Using Loops\n2. Using Recursion\n");
    scanf("%d", &choice);
    if (choice == 1) {
        generatePascalLoop(n);
    } else if (choice == 2) {
        generatePascalRecursive(n);
    } else {
        printf("Invalid choice.\n");
    }
    return 0;
}
```

Lab Challenge 3: Number Guessing Game

Write a C program that implements a simple number guessing game. The program should generate a random number between 1 and 100, and the user should guess the number within a limited number of attempts.

Challenge: -Provide hints to the user if the guessed number is too high or too low

Answer :-

```
#include <stdio.h>

#include <stdlib.h>
```

```
#include <time.h>

#define MAX_ATTEMPTS 7

int main() {
   int randomNumber, guess, attempts = 0;
   char playAgain;
   srand(time(0));
   do {
      randomNumber = (rand() % 100) + 1;
      attempts = 0;


      printf("\nWelcome to the Number Guessing Game!\n");
      printf("I have chosen a number between 1 and 100. Can you guess it?\n");
      printf("You have %d attempts.\n", MAX_ATTEMPTS);
      while (attempts < MAX_ATTEMPTS) {
         printf("\nEnter your guess: ");
         scanf("%d", &guess);


         attempts++;


         if (guess == randomNumber) {
            printf("Congratulations! You guessed the correct number %d in %d attempt(s).\n",
randomNumber, attempts);
            break;
         } else if (guess > randomNumber) {
            printf("Your guess is too high. Try again!\n");
         } else {
```

```
            printf("Your guess is too low. Try again!\n");
        }

        // Check if attempts are exhausted
        if (attempts == MAX_ATTEMPTS) {
            printf("\nYou've used all your attempts. The correct number was %d.\n", randomNumber);
        }
    }
    printf("\nDo you want to play again? (y/n): ");
    scanf(" %c", &playAgain);

} while (playAgain == 'y' || playAgain == 'Y');

printf("Thank you for playing the Number Guessing Game! Goodbye!\n");
return 0;
}
```