Module 2 – Introduction to Programming

Name : VAIDANGI GADHIYA

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.

C programming language was developed in 1972 by Dennis Ritchie at Bell Labs. It was mainly created to develop the Unix operating system.
C became popular because it is simple, fast, and powerful. It allows programmers to work closely with computer hardware and manage memory efficiently. Over time, C was improved and standardized, which made it more reliable and portable.
C is still used today because:

- It is very fast and efficient.
- Many modern languages are based on C.
- It is widely used in system programming and embedded systems.
- It helps students understand programming basics clearly.

• 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.

Three Real-World Applications of C Programming

Operating Systems
C is widely used in developing operating systems:

- Linux
- Microsoft Windows (partially written in C)
- Unix

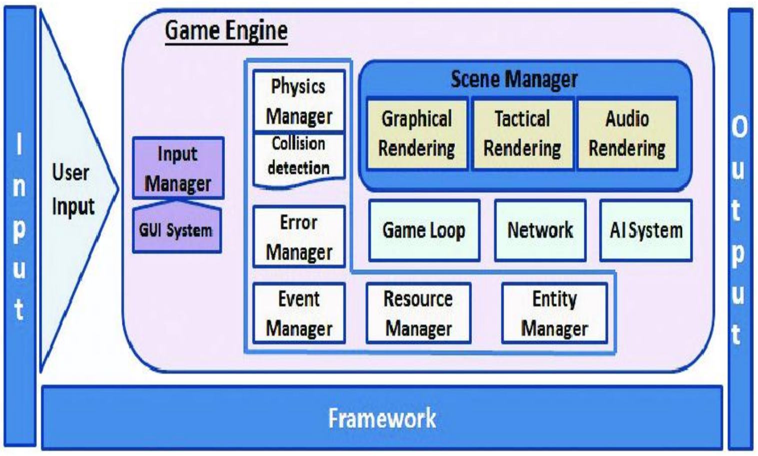
C is preferred because it provides direct access to hardware and memory.

Embedded Systems
C is extensively used in embedded systems like:

- Microcontrollers
- IoT devices
- Home appliances
- Automotive systems

Embedded systems require fast execution and low memory usage, which C provides.

Game Development



C is used in game engines and performance-critical components.

- Doom was originally written in C.
- Many game engines use C for speed and hardware control.

Game development needs high speed and graphics performance, and C helps achieve that.

C programming has a rich history and strong foundation in system-level programming. It remains relevant because of its speed, efficiency, and flexibility. From operating systems to embedded systems and games, C continues to play a vital role in the software industry.

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.

To run C programs, we need a C compiler and an IDE.

Steps to Install C Compiler (GCC):

1. Download and install GCC (MinGW for Windows).
2. Add it to system PATH.
3. Open Command Prompt and type:
4. gcc --version

If version appears, installation is successful.

Install IDE:

You can use:

- Dev-C++
- Visual Studio Code
- Code::Blocks

Steps:

1. Download and install IDE.
2. Create a new C file (.c).
3. Write program and run it.

• 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.

1) Installed C Compiler and Configured IDE

- Installed GCC (MinGW).
- Checked installation using:
- gcc --version

- Installed an IDE like Dev-C++ or Visual Studio Code.
- Created a new file named hello.c.

2) First Program – Hello World

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

3) Compile and Run

Compile:

```
gcc hello.c -o hello
```

Run:

```
hello
```

Output:

```
Hello, World
```

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.

The basic structure of a C program includes the following parts:

1) Header Files

Header files are included using #include.

Example:

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

It allows us to use functions like printf().

2) Main Function

Execution of a C program starts from the main() function.

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

3) Comments

Comments explain the code and are not executed.

- Single-line comment:
- // This is a comment
- Multi-line comment:
- /* This is
-    a multi-line comment */

4) Data Types

Data types define the type of data a variable can store.

- int → Integer values (10, 20)

- float → Decimal values (10.5)
- char → Single character ('A')

5) Variables

Variables are used to store data.

Example:

int age = 18;

float marks = 85.5;

char grade = 'A';

• 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.

LAB EXERCISE

C Program Using Variables, Constants, and Comments

```
#include <stdio.h>
int main() {
   int age = 20;
   float height = 5.6;
   char grade = 'A';

   const int year = 2026;

   printf("Age: %d\n", age);
   printf("Height: %.1f\n", height);
   printf("Grade: %c\n", grade);
   printf("Year: %d\n", year);

   return 0;
}
```

Output:

```
Age: 20
Height: 5.6
Grade: A
Year: 2026
```

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.

1) Arithmetic Operators

Used for mathematical calculations.

+ (Addition)

- (Subtraction)

* (Multiplication)

/ (Division)

% (Modulus)

Example:

int a = 10 + 5;

2) Relational Operators

Used to compare two values. Result is true (1) or false (0).

== Equal to

!= Not equal to
> Greater than
< Less than
>= Greater than or equal
<= Less than or equal
Example:
a > b

3) Logical Operators
Used to combine conditions.
&& (AND)
|| (OR)
! (NOT)
Example:
a > 5 && b < 10

4) Assignment Operators
Used to assign values.
=
+=
-=
*=
/=
Example:
a += 5;

5) Increment / Decrement Operators
Used to increase or decrease value by 1.
++ (Increment)
-- (Decrement)
Example:
a++;

6) Bitwise Operators
Work on binary values.
& (AND)
| (OR)
^ (XOR)
~ (NOT)
<< (Left shift)
>> (Right shift)

7) Conditional (Ternary) Operator
Used as shorthand for if-else.
Syntax:
condition ? expression1 : expression2;
Example:
max = (a > b) ? a : b;

• 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.

```
#include <stdio.h>

int main() {

   int a, b;

   printf("Enter two numbers: ");
   scanf("%d %d", &a, &b);

   printf("\nAddition: %d\n", a + b);
   printf("Subtraction: %d\n", a - b);
   printf("Multiplication: %d\n", a * b);
   printf("Division: %d\n", a / b);

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

```
    printf("\n(a > 0 && b > 0): %d\n", (a > 0 && b > 0));
    printf("(a > 0 || b > 0): %d\n", (a > 0 || b > 0));

    return 0;
}
```

Output:

```
Enter two numbers: 10 5
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2
a > b: 1
a == b: 0
(a > 0 && b > 0): 1
(a > 0 || b > 0): 1
```

Control Flow Statements in C

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

1) if Statement

Used to execute a block of code if a condition is true.

Example:

```
if (a > 0) {
    printf("Positive number");
}
```

2) if-else Statement

Used when there are two conditions (true or false).

Example:

```
if (a % 2 == 0) {
    printf("Even");
} else {
    printf("Odd");
}
```

3) Nested if-else

An if-else inside another if-else.

Used when multiple conditions are checked.

Example:

```
if (a > 0) {
    if (a % 2 == 0)
```

```
        printf("Positive Even");
    else
        printf("Positive Odd");
}
```

4) switch Statement

Used to select one option from many choices.

Example:

```
switch(choice) {
    case 1: printf("One"); break;
    case 2: printf("Two"); break;
    default: printf("Invalid");
}
```

• 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.).

```
#include <stdio.h>

int main() {

    int num, month;

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

    if (num % 2 == 0)
        printf("Number is Even\n");
    else
        printf("Number is Odd\n");

    printf("\nEnter month number (1-12): ");
    scanf("%d", &month);

    switch(month) {
        case 1: printf("January"); break;
        case 2: printf("February"); break;
        case 3: printf("March"); break;
        case 4: printf("April"); break;
        case 5: printf("May"); break;
        case 6: printf("June"); break;
        case 7: printf("July"); break;
        case 8: printf("August"); break;
        case 9: printf("September"); break;
        case 10: printf("October"); break;
        case 11: printf("November"); break;
        case 12: printf("December"); break;
        default: printf("Invalid Month");
    }

    return 0;
}
```

Output:

```
Enter a number: 6
Number is Even

Enter month number (1-12): 3
March
```

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.

1) while Loop

- Checks condition first, then executes the loop.
- Used when number of iterations is not fixed.

Syntax:

```
while(condition)
{
}
```

2) for Loop

- Used when number of iterations is known.
- Initialization, condition, and increment are written in one line.

Syntax:

```
for(initialization; condition; increment)
{

}
```

3) do-while Loop

- Executes at least one time because condition is checked after execution.
- Used when loop must run at least once.

Syntax:

```
do {
  } while(condition);
```

Comparison

Loop	**Condition Check**	**Best Used When**
while	Before loop	Iterations unknown
for	Before loop	Iterations known
do-while	After loop	Execute at least once

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

```
#include <stdio.h>

int main() {
    int i;

   printf("Using while loop:\n");
   i = 1;
   while(i <= 10) {
      printf("%d ", i);
      i++;
   }

   printf("\n\nUsing for loop:\n");
   for(i = 1; i <= 10; i++) {
      printf("%d ", i);
   }

   printf("\n\nUsing do-while loop:\n");
   i = 1;
   do {
      printf("%d ", i);
      i++;
   } while(i <= 10);

   return 0;
}
```

Output :

```
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
```

Loop Control Statements

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

1) break Statement

Used to immediately exit from a loop or switch statement.

Example:

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

(Output stops at 4)

2) continue Statement

Used to skip the current iteration and move to the next iteration of the loop.

Example:

```
for(int i = 1; i <= 5; i++) {
   if(i == 3)
      continue;
   printf("%d ", i);
}
```

(Number 3 is skipped)

3) goto Statement

Used to transfer control to another part of the program using a label.

It is rarely used because it can make code complex.

Example:

```
goto label;
printf("This will be skipped");
label:
printf("Jumped using 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.

```
#include <stdio.h>

int main() {

   int i;
```

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

    printf("\n\nUsing continue:\n");
    for(i = 1; i <= 5; i++) {
        if(i == 3)
            continue;
        printf("%d ", i);
    }

    return 0;
}
```

Output:

```
Using break:
1 2 3 4

Using continue:
1 2 4 5
```

Functions in C

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

Functions in C – Theory

What are Functions in C?

In C, a function is a block of code that performs a specific task.
Functions help in dividing a large program into smaller, manageable parts. This makes the program easy to understand, test, and maintain.

Functions improve:

- Code reusability
- Modularity
- Readability
- Debugging

Types of Functions in C

1. **Library Functions**
   Predefined functions provided by C.
   Example: printf(), scanf(), sqrt()
2. **User-defined Functions**
   Functions created by the programmer to perform specific tasks.

Parts of a Function

There are three main parts of a function:

Function Declaration (Prototype)

It tells the compiler about:

- Function name
- Return type
- Number and type of parameters

Syntax:

```
return_type function_name(parameters);
```

Example:

```
int add(int, int);
```

Function Definition

It contains the actual code that performs the task.

Syntax:

```
return_type function_name(parameters)
{
}
```

Example:

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

Function Call

It is used to execute the function.

Example:

```
int result = add(5, 3);
```

Advantages of Functions

- Reduces code repetition
- Makes program structured
- Easier to test and debug
- Saves memory by reusing code

- Improves program clarity

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

```
#include <stdio.h>

int factorial(int);

int main()
{
   int num, result;

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

   result = factorial(num);

   printf("Factorial of %d = %d", num, result);

   return 0;
}

int factorial(int n)
{
   int i, fact = 1;

   for(i = 1; i <= n; i++)
   {
      fact = fact * i;
   }

   return fact;
}
```

Arrays in C

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

Concept of Arrays in C

an array is a collection of elements of the same data type stored in continuous memory locations.

It allows multiple values to be stored under one variable name and accessed using an index (starting from 0).

Syntax:

data_type array_name[size];

Example:

```
int arr[5];
```

One-Dimensional Array (1D Array)

- Stores elements in a single row.
- Uses one index.
- Suitable for storing lists of values.

Example:

```
int arr[5] = {10, 20, 30, 40, 50};

printf("%d", arr[2]);   // Output: 30
```

Multi-Dimensional Array (2D Array)

- Stores elements in rows and columns.
- Uses two or more indexes.
- Commonly used for matrices and tables.

Syntax:

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

Example:

```
int matrix[2][2] = {
   {1, 2},
   {3, 4}
};
printf("%d", matrix[1][0]);   // Output: 3
```

Difference Between 1D and Multi-Dimensional Array

Feature	One-Dimensional Array	Multi-Dimensional Array
Indexes	One index	Two or more indexes
Structure	Single row	Rows & Columns
Example	int a[5];	int a[3][3];
Usage	List of values	Matrix / Table

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.

```c
#include <stdio.h>

int main()
{
   int a[5];
   int i;

   printf("Enter 5 numbers:\n");
   for(i=0; i<5; i++)
   {
      scanf("%d",&a[i]);
   }

   printf("Numbers are:\n");
   for(i=0; i<5; i++)
   {
      printf("%d ",a[i]);
   }


   int b[3][3];
   int j, sum=0;

   printf("\nEnter 3x3 matrix elements:\n");

   for(i=0; i<3; i++)
   {
      for(j=0; j<3; j++)
      {
         scanf("%d",&b[i][j]);
      }
   }

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

   printf("Sum of all elements = %d", sum);

   return 0;
}
```

Output :

```
Enter 5 numbers:
1 2 3 4 5
Numbers are:
1 2 3 4 5
```

```
Enter 3x3 matrix elements:
1 2 3
4 5 6
7 8 9
Sum of all elements = 45
```

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?

What are Pointers?

A pointer is a variable that stores the **address of another variable**.

In C, pointers are used to access and modify data using memory addresses.

Declaration of Pointer

int *ptr;

Here, ptr is a pointer to an integer.

Initialization of Pointer

int a = 10;

int *ptr = &a;

&a gives the address of variable a.

Now ptr stores the address of a.

Why Pointers are Important?

- They allow direct memory access.
- Used in functions (call by reference).
- Used in arrays and strings.
- Required for dynamic memory allocation.
- Helpful in structures and data structures like linked list.

• 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.

```
#include <stdio.h>

int main()
{
   int a = 10;
   int *p;

   p = &a;

   printf("Value of a before change = %d\n", a);

   *p = 20;

   printf("Value of a after change = %d\n", a);
```

```
    return 0;
}
```

Output:

```
Value of a before change = 10

Value of a after change = 20
```

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.

a string is a collection of characters ending with a **null character (\0)**.
String functions are available in the header file #include <string.h>.

1. strlen()

- Used to find the length of a string.
- Returns number of characters (excluding \0).

Example:
strlen("Hello");   // Output: 5
Useful when we need to count characters in a string.

2. strcpy()

- Used to copy one string into another.

Example:
strcpy(str2, str1);
Useful for copying data from one string variable to another.

3. strcat()

- Used to join (concatenate) two strings.

Example:
strcat(str1, str2);
Useful when combining two names or words.

4. strcmp()

- Used to compare two strings.
- Returns 0 if strings are equal.

Example:
strcmp(str1, str2);
Useful in password checking or comparing names.

5. strchr()

- Used to find the first occurrence of a character in a string.

Example:
strchr(str, 'a');
Useful for searching a character in a string.

• 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().

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

int main()
{
   char str1[50], str2[50];

   printf("Enter first string: ");
   gets(str1);

   printf("Enter second string: ");
   gets(str2);

   strcat(str1, str2);

   printf("Concatenated string = %s\n", str1);

   printf("Length of string = %lu", strlen(str1));

   return 0;
}
```

Output :

```
Enter first string: Hello

Enter second string: World

Concatenated string = HelloWorld

Length of string = 10
```

Structures in C

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

What is a Structure?

A structure in C is a user-defined data type that allows us to store different types of data under one name. For example, a student has:

- Name (string)
- Roll number (int)
- Marks (float)

Instead of creating separate variables, we can group them using a structure.

Declaration of Structure

```
struct student
{
   char name[50];
   int roll;
   float marks;
};
```
Initialization of Structure

```
struct student s1 = {"Rahul", 101, 85.5};
```
Accessing Structure Members

We use the **dot (.) operator** to access members.

Example:
```
printf("%s", s1.name);
printf("%d", s1.roll);
printf("%.2f", s1.marks);
```
Why Structures are Important?

- To store related data together
- Used in real-world applications
- Helpful in databases and records
- Used with arrays and pointers

• 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

```
#include <stdio.h>

struct student
{
   char name[50];
   int roll;
   float marks;
};

int main()
{
   struct student s[3];
   int i;

   for(i=0; i<3; i++)
   {
      printf("Enter details of student %d\n", i+1);

      printf("Name: ");
      scanf("%s", s[i].name);

      printf("Roll No: ");
      scanf("%d", &s[i].roll);

      printf("Marks: ");
      scanf("%f", &s[i].marks);
```

```
    }

    printf("\nStudent Details:\n");

    for(i=0; i<3; i++)
    {
        printf("\nStudent %d\n", i+1);
        printf("Name: %s\n", s[i].name);
        printf("Roll No: %d\n", s[i].roll);
        printf("Marks: %.2f\n", s[i].marks);
    }

    return 0;
}
```

Output

```
Enter details of student 1

Name: Riya

Roll No: 1

Marks: 85


Enter details of student 2

Name: Amit

Roll No: 2

Marks: 90


Enter details of student 3

Name: Neha

Roll No: 3

Marks: 88


Student Details:


Student 1

Name: Riya

Roll No: 1

Marks: 85.00

...
```

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.

Importance of File Handling
File handling in C is important because it allows us to:

- Store data permanently in files
- Read data later when needed
- Handle large amount of data
- Maintain records (like student details, employee data, etc.)

Without file handling, data is lost after program execution.
File Operations in C
1. Opening a File
We use fopen() function.
FILE *fp;
fp = fopen("file.txt", "w");
Modes:

- "r" → Read
- "w" → Write
- "a" → Append

2. Writing to a File
We use fprintf(), fputs(), or fputc().
Example:
fprintf(fp, "Hello");
3. Reading from a File
We use fscanf(), fgets(), or fgetc().
Example:
fgets(str, 50, fp);
4. Closing a File
We use fclose() function.
fclose(fp);
Closing a file is important to save changes and free memory.

• 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.

```
#include <stdio.h>

int main()
{
   FILE *f;
   char s[50];

   f = fopen("file.txt","w");
   printf("Enter text: ");
   scanf("%s", s);
   fprintf(f,"%s", s);
   fclose(f);

   f = fopen("file.txt","r");
   fscanf(f,"%s", s);
```

```
    printf("Data in file: %s", s);
    fclose(f);

    return 0;
}
```

Output

```
Enter text: Hello

Data in file: Hello
```

EXTRA LAB EXERCISES FOR IMPROVING PROGRAMMING LOGIC

Operators LAB EXERCISE 1: Simple Calculator

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

```
#include <stdio.h>

int main()
{
    int a, b;
    char op;

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

    printf("Enter second number: ");
    scanf("%d", &b);

    printf("Enter operator (+, -, *, /, %%): ");
    scanf(" %c", &op);

    if(op == '+')
        printf("Result = %d", a + b);

    else if(op == '-')
        printf("Result = %d", a - b);

    else if(op == '*')
        printf("Result = %d", a * b);

    else if(op == '/')
        printf("Result = %d", a / b);

    else if(op == '%')
```

```
        printf("Result = %d", a % b);

    else
        printf("Invalid Operator");

    return 0;
}
```

Output

```
Enter first number: 10

Enter second number: 5

Enter operator (+, -, *, /, %): +

Result = 15

Enter first number: 10

Enter second number: 5

Enter operator (+, -, *, /, %): $

Invalid Operator
```

LAB EXERCISE 2: Check Number Properties

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

o Whether the number is even or odd.

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

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

```
#include <stdio.h>

int main()
{
    int n;

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

    if(n % 2 == 0)
        printf("Number is Even\n");
    else
        printf("Number is Odd\n");

    if(n > 0)
        printf("Number is Positive\n");
    else if(n < 0)
        printf("Number is Negative\n");
    else
        printf("Number is Zero\n");
    if(n % 3 == 0 && n % 5 == 0)
        printf("Number is multiple of 3 and 5\n");
    else
        printf("Number is not multiple of 3 and 5\n");

    return 0;
}
```

Output:

```
Enter a number: 15

Number is Odd

Number is Positive

Number is multiple of 3 and 5
```

2. Control Statements

LAB EXERCISE 1: Grade Calculator

Write a C program that takes marks as input and displays grade:

- **Marks > 90 → Grade A**
- **Marks > 75 and <= 90 → Grade B**
- **Marks > 50 and <= 75 → Grade C**
- **Marks <= 50 → Grade D**

Use if-else or switch statements for the decision-making process.

```
#include <stdio.h>

int main()
{
    int marks;

    printf("Enter marks: ");
    scanf("%d",&marks);

    if(marks > 90)
        printf("Grade A");
    else if(marks > 75)
        printf("Grade B");
    else if(marks > 50)
        printf("Grade C");
    else
        printf("Grade D");

    return 0;
}
```

Output:

```
Enter marks: 82
```

Grade B

LAB EXERCISE 2: Number Comparison
Write a C program that takes three numbers and finds:

- **The largest number**
- **The smallest number**
- **Challenge: Solve the problem using both if-else and switch-case statements.**

Program 1 (Using if-else):

```
#include <stdio.h>

int main()
{
   int a,b,c,max,min;

   printf("Enter 3 numbers: ");
   scanf("%d%d%d",&a,&b,&c);

   max = a;
   if(b > max) max = b;
   if(c > max) max = c;

   min = a;
   if(b < min) min = b;
   if(c < min) min = c;

   printf("Largest = %d\n",max);
   printf("Smallest = %d",min);

   return 0;
}
```

Program 2 (Using switch-case – simple version)

```
#include <stdio.h>

int main()
{
   int a,b,c;

   printf("Enter 3 numbers: ");
   scanf("%d%d%d",&a,&b,&c);

   switch(a>b && a>c)
   {
      case 1: printf("Largest = %d\n",a); break;
      default:
         switch(b>c)
         {
            case 1: printf("Largest = %d\n",b); break;
            default: printf("Largest = %d\n",c);
         }
   }
   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**

```
#include <stdio.h>

int main() {
   int marks;

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

   if (marks > 90) {
      printf("Grade A");
   }
   else if (marks > 75 && marks <= 90) {
      printf("Grade B");
   }
   else if (marks > 50 && marks <= 75) {
      printf("Grade C");
   }
   else {
      printf("Grade D");
   }

   return 0;
}
```

• Use if-else or switch statements for the decision-making process.

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.**

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

Method 1: Using if-else

```
#include <stdio.h>

int main() {
   int a, b, c;
   int largest, smallest;

   printf("Enter three numbers: ");
   scanf("%d %d %d", &a, &b, &c);

   if (a >= b && a >= c)
```

```
        largest = a;
    else if (b >= a && b >= c)
        largest = b;
    else
        largest = c;

    if (a <= b && a <= c)
        smallest = a;
    else if (b <= a && b <= c)
        smallest = b;
    else
        smallest = c;

    printf("Largest number = %d\n", largest);
    printf("Smallest number = %d\n", smallest);

    return 0;
}
```

Method 2: Using switch-case

```
#include <stdio.h>

int main() {
    int a, b, c;
    int largest, smallest;
    int choice;

    printf("Enter three numbers: ");
    scanf("%d %d %d", &a, &b, &c);

    choice = (a >= b && a >= c) ? 1 :
             (b >= a && b >= c) ? 2 : 3;

    switch(choice) {
        case 1: largest = a; break;
        case 2: largest = b; break;
        case 3: largest = c; break;
    }

    choice = (a <= b && a <= c) ? 1 :
             (b <= a && b <= c) ? 2 : 3;

    switch(choice) {
        case 1: smallest = a; break;
        case 2: smallest = b; break;
        case 3: smallest = c; break;
    }

    printf("Largest number = %d\n", largest);
    printf("Smallest number = %d\n", smallest);
    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.

```
#include <stdio.h>

int main() {
    int num, i, flag = 0;

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

    if (num <= 1) {
        printf("Not a Prime Number");
    }
    else {
        for (i = 2; i <= num / 2; i++) {
            if (num % i == 0) {
                flag = 1;
                break;
            }
        }

        if (flag == 0)
            printf("Prime Number");
        else
            printf("Not a Prime Number");
    }

    return 0;
}
```

• Challenge: Modify the program to print all prime numbers between 1 and a given number.

```
#include <stdio.h>

int main() {
    int n, i, j, flag;

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

    printf("Prime numbers between 1 and %d are:\n", n);

    for (i = 2; i <= n; i++) {
        flag = 0;

        for (j = 2; j <= i / 2; j++) {
            if (i % j == 0) {
                flag = 1;
                break;
            }
        }

        if (flag == 0)
            printf("%d ", i);
    }

    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.

```
#include <stdio.h>

int main() {
    int num, i;

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

    for (i = 1; i <= 10; i++) {
        printf("%d x %d = %d\n", num, i, num * i);
    }

    return 0;
}
```

• Challenge: Allow the user to input the range of the multiplication table (e.g., from 1 to N).

```
#include <stdio.h>

int main() {
    int num, range, i;

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

    printf("Enter the range: ");
    scanf("%d", &range);

    for (i = 1; i <= range; 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.

```
#include <stdio.h>

int main() {
    int num, digit, sum = 0;
    printf("Enter an integer: ");
    scanf("%d", &num);

    while (num != 0) {
        digit = num % 10;   // Get last digit
        sum = sum + digit;  // Add digit to sum
        num = num / 10;     // Remove last digit
    }

    printf("Sum of digits = %d", sum);

    return 0;
}
```

• Challenge: Extend the program to reverse the digits of the number.

```
#include <stdio.h>

int main() {
    int num, digit, reverse = 0;

    printf("Enter an integer: ");
    scanf("%d", &num);

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

    printf("Reversed number = %d", 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.

```
#include <stdio.h>

int main() {
    int arr[10], i;
    int max, min;

    printf("Enter 10 integers:\n");
    for(i = 0; i < 10; i++) {
        scanf("%d", &arr[i]);
    }

    max = 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 = %d\n", max);
    printf("Minimum = %d\n", min);

    return 0;
}
```
• Challenge: Extend the program to sort the array in ascending order.
```
#include <stdio.h>

int main() {
    int arr[10], i, j, temp;

    printf("Enter 10 integers:\n");
    for(i = 0; i < 10; i++) {
        scanf("%d", &arr[i]);
    }

    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("Array in ascending order:\n");
    for(i = 0; i < 10; i++) {
        printf("%d ", arr[i]);
    }

    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.
```
#include <stdio.h>

int main() {
    int a[2][2], b[2][2], sum[2][2];
    int i, j;

    printf("Enter elements of first matrix:\n");
    for(i = 0; i < 2; i++)
```

```
        for(j = 0; j < 2; j++)
            scanf("%d", &a[i][j]);

    printf("Enter elements of second matrix:\n");
    for(i = 0; i < 2; i++)
        for(j = 0; j < 2; j++)
            scanf("%d", &b[i][j]);

    for(i = 0; i < 2; i++)
        for(j = 0; j < 2; j++)
            sum[i][j] = a[i][j] + b[i][j];

    printf("Resultant Matrix:\n");
    for(i = 0; i < 2; i++) {
        for(j = 0; j < 2; j++)
            printf("%d ", sum[i][j]);
        printf("\n");
    }

    return 0;
}
```

• **Challenge: Extend the program to work with 3x3 matrices and matrix multiplication.**

```
#include <stdio.h>

int main() {
    int a[3][3], b[3][3], result[3][3];
    int i, j, k;

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

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

    for(i = 0; i < 3; i++) {
        for(j = 0; j < 3; j++) {
            result[i][j] = 0;
            for(k = 0; k < 3; k++) {
                result[i][j] += a[i][k] * b[k][j];
            }
        }
    }

    printf("Resultant Matrix after Multiplication:\n");
    for(i = 0; i < 3; i++) {
        for(j = 0; j < 3; j++)
            printf("%d ", result[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.

```
#include <stdio.h>

int main() {
    int n, i;
    int arr[100];
    int sum = 0;

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

    printf("Enter %d numbers:\n", n);
    for(i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
        sum += arr[i];
    }

    printf("Sum = %d", sum);

    return 0;
}
```
• Challenge: Modify the program to also find the average of the numbers.

```
#include <stdio.h>

int main() {
    int n, i;
    int arr[100];
    int sum = 0;
    float average;

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

    printf("Enter %d numbers:\n", n);
    for(i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
        sum += arr[i];
    }

    average = (float)sum / n;

    printf("Sum = %d\n", sum);
    printf("Average = %.2f", 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.

```
#include <stdio.h>

int fibonacci(int n) {
   if (n == 0)
      return 0;
   else if (n == 1)
      return 1;
   else
      return fibonacci(n - 1) + fibonacci(n - 2);
}

int main() {
   int n, i;

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

   printf("Fibonacci Series:\n");
   for (i = 0; i < n; i++) {
      printf("%d ", fibonacci(i));
   }

   return 0;
}
```

• Challenge: Modify the program to calculate the Nth Fibonacci number using both iterative and recursive methods. Compare their efficiency.

Iterative Method

```
int fib_iterative(int n) {
   int a = 0, b = 1, c, i;

   if (n == 0) return 0;

   for (i = 2; i <= n; i++) {
      c = a + b;
      a = b;
      b = c;
   }
   return b;
}
```

Recursive Method

```
int fib_recursive(int n) {
   if (n <= 1)
      return n;
   return fib_recursive(n - 1) + fib_recursive(n - 2);
}
```

LAB EXERCISE 2: Factorial Calculation

• Write a C program that calculates the factorial of a given number using a function.

```
#include <stdio.h>

long long factorial(int n) {
  long long fact = 1;
  int i;

  for (i = 1; i <= n; i++) {
    fact *= i;
  }
  return fact;
}

int main() {
  int num;

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

  printf("Factorial = %lld", factorial(num));

  return 0;
}
```

• Challenge: Implement both an iterative and a recursive version of the factorial function and compare their performance for large numbers.

```
#include <stdio.h>

// Iterative Method
long long fact_iterative(int n) {
  long long fact = 1;
  int i;

  for(i = 1; i <= n; i++) {
    fact = fact * i;
  }
  return fact;
}

// Recursive Method
long long fact_recursive(int n) {
  if(n == 0 || n == 1)
    return 1;
  else
    return n * fact_recursive(n - 1);
}

int main() {
  int num;

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

  printf("Iterative Factorial = %lld\n", fact_iterative(num));
```

```
    printf("Recursive Factorial = %lld\n", fact_recursive(num));

    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.**

```
#include <stdio.h>

int isPalindrome(int num) {
    int original = num, reverse = 0, digit;

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

    if (original == reverse)
        return 1;
    else
        return 0;
}

int main() {
    int num;

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

    if (isPalindrome(num))
        printf("Palindrome Number");
    else
        printf("Not a Palindrome Number");

    return 0;
}
```

- **Challenge: Modify the program to check if a given string is a palindrome.**

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

int main() {
    char str[100];
    int i, length, flag = 1;

    printf("Enter a string: ");
    scanf("%s", str);

    length = strlen(str);

    for (i = 0; i < length / 2; i++) {
        if (str[i] != str[length - i - 1]) {
            flag = 0;
            break;
        }
    }

    if (flag)
        printf("Palindrome String");
    else
        printf("Not a Palindrome String");

    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.

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

void reverseString(char str[]) {
    int i, length;
    char temp;

    length = strlen(str);

    for(i = 0; i < length / 2; i++) {
        temp = str[i];
        str[i] = str[length - i - 1];
        str[length - i - 1] = temp;
    }
}

int main() {
    char str[100];

    printf("Enter a string: ");
    scanf("%s", str);

    reverseString(str);

    printf("Reversed String = %s", str);

    return 0;
}
```

• Challenge: Write the program without using built-in string handling functions.

```
#include <stdio.h>

void reverseString(char str[]) {
    int i = 0, length = 0;
    char temp;

    // Find length manually
    while(str[length] != '\0') {
        length++;
    }

    for(i = 0; i < length / 2; i++) {
        temp = str[i];
        str[i] = str[length - i - 1];
        str[length - i - 1] = temp;
    }
}

int main() {
    char str[100];

    printf("Enter a string: ");
    scanf("%s", str);

    reverseString(str);

    printf("Reversed String = %s", 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.

```
#include <stdio.h>

int main() {
    char str[100];
    int i = 0;
    int vowels = 0, consonants = 0;

    printf("Enter a string: ");
    scanf("%s", str);

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

        if(str[i]=='a'||str[i]=='e'||str[i]=='i'||str[i]=='o'||str[i]=='u'||
           str[i]=='A'||str[i]=='E'||str[i]=='I'||str[i]=='O'||str[i]=='U') {
            vowels++;
        }
        else if((str[i] >= 'a' && str[i] <= 'z') ||
                (str[i] >= 'A' && str[i] <= 'Z')) {
            consonants++;
        }

        i++;
    }

    printf("Vowels = %d\n", vowels);
    printf("Consonants = %d\n", consonants);
    return 0;
}
```
• Challenge: Extend the program to also count digits and special characters.
```
#include <stdio.h>

int main() {
    char str[100];
    int i = 0;
    int vowels = 0, consonants = 0, digits = 0, special = 0;

    printf("Enter a string: ");
    scanf("%s", str);

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

        if(str[i]=='a'||str[i]=='e'||str[i]=='i'||str[i]=='o'||str[i]=='u'||
           str[i]=='A'||str[i]=='E'||str[i]=='I'||str[i]=='O'||str[i]=='U') {
            vowels++;
        }
        else if((str[i] >= 'a' && str[i] <= 'z') ||
                (str[i] >= 'A' && str[i] <= 'Z')) {
            consonants++;
        }
        else if(str[i] >= '0' && str[i] <= '9') {
            digits++;
        }
        else {
            special++;
        }

        i++;
    }

    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.
```
#include <stdio.h>

int main() {
    char str[200];
    int i = 0, words = 0;

    printf("Enter a sentence: ");
    fgets(str, sizeof(str), stdin);
    // Count words
    while(str[i] != '\0') {
        if(str[i] == ' ' && str[i+1] != ' ' && str[i+1] != '\n') {
            words++;
        }
        i++;
    }

    // If sentence is not empty, add 1 word
    if(str[0] != '\n')
        words++;

    printf("Number of words = %d", words);

    return 0;
}
```

• **Challenge: Modify the program to find the longest word in the sentence.**

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

int main() {
    char str[200], longest[50];
    int i = 0, j = 0, maxLength = 0, length = 0;
    char temp[50];

    printf("Enter a sentence: ");
    fgets(str, sizeof(str), stdin);

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

        if(str[i] != ' ' && str[i] != '\n') {
            temp[j++] = str[i];
            length++;
        } else {
            temp[j] = '\0';

            if(length > maxLength) {
                maxLength = length;
                strcpy(longest, temp);
            }

            length = 0;
            j = 0;
        }
        i++;
    }

    printf("Longest word = %s", longest);

    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).

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

int main() {
   int num, original, remainder, result = 0, digits = 0;
   int temp;

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

   original = num;
   temp = num;

   // Count number of digits
   while(temp != 0) {
      temp = temp / 10;
      digits++;
   }

   temp = num;

   // Calculate Armstrong sum
   while(temp != 0) {
      remainder = temp % 10;
      result += pow(remainder, digits);
      temp = temp / 10;
   }

   if(result == original)
      printf("It is an Armstrong number");
   else
      printf("It is not an Armstrong number");

   return 0;
}
```

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

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

int main() {
   int num, temp, remainder, result, digits;

   printf("Armstrong numbers between 1 and 1000 are:\n");

   for(num = 1; num <= 1000; num++) {

      temp = num;
```

```
        result = 0;
        digits = 0;

        // Count digits
        int t = num;
        while(t != 0) {
            t = t / 10;
            digits++;
        }

        // Calculate Armstrong sum
        while(temp != 0) {
            remainder = temp % 10;
            result += pow(remainder, digits);
            temp = temp / 10;
        }

        if(result == num)
            printf("%d ", num);
    }

    return 0;
}
```

Lab Challenge 2: Pascal’s Triangle

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

```
#include <stdio.h>

int main() {
    int n, i, j, space;
    int coef = 1;

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

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

        // Print spaces
        for(space = 1; space <= n - i; space++)
            printf(" ");

        coef = 1;

        for(j = 0; j <= i; j++) {
            if(j == 0 || i == 0)
                coef = 1;
            else
                coef = coef * (i - j + 1) / j;

            printf("%d ", coef);
        }
        printf("\n");
    }
    return 0;
}
```

• Challenge: Implement the same program using a recursive function.

```
#include <stdio.h>

int combination(int n, int r) {
    if(r == 0 || r == n)
        return 1;
    else
        return combination(n-1, r-1) + combination(n-1, r);
}

int main() {
    int n, i, j, space;

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

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

        for(space = 1; space <= n - i; space++)
            printf(" ");

        for(j = 0; j <= i; j++) {
            printf("%d ", combination(i, j));
        }

        printf("\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.

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

int main() {
    int number, guess, attempts = 0;

    srand(time(0));
    number = rand() % 100 + 1;

    printf("Guess the number (1 to 100)\n");
    printf("You have 5 attempts.\n");

    while(attempts < 5) {
        printf("Enter your guess: ");
        scanf("%d", &guess);

        attempts++;

        if(guess == number) {
            printf("Correct! You guessed it.\n");
            break;
        }
        else if(guess > number) {
            printf("Too High!\n");
        }
        else {
            printf("Too Low!\n");
        }
    }

    if(guess != number) {
        printf("Game Over! The number was %d\n", number);
    }

    return 0;
}
```

• **Challenge: Provide hints to the user if the guessed number is too high or too low.**

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

int main() {
    int number, guess;

    srand(time(0));
    number = rand() % 100 + 1;

    printf("Guess the number between 1 and 100\n");

    while(1) {
        printf("Enter your guess: ");
        scanf("%d", &guess);

        if(guess == number) {
            printf("Correct! You guessed the number.\n");
            break;
        }
        else if(guess > number) {
            printf("Hint: Too High! Try smaller number.\n");
        }
        else {
            printf("Hint: Too Low! Try bigger number.\n");
        }
    }

    return 0;
}
```