Module 2 – Introduction to Programming

Q-1-TE: Write an essay covering the history and evolution of C programming. Explain its importance and why it is still used today.

- C programming language, created by Dennis Ritchie in the early 1970s at Bell Labs.
- Its efficiency, simplicity, and flexibility have made it foundational to many programming languages and critical applications.
- C was developed for the Unix operating system to replace assembly language, combining low-level capabilities with structured programming features.
- its viability for system-level programming.

  Evolution:

  - K&R C (1978): Dennis Ritchie and Brian Kernighan's book standardized early C features.
  - ANSI C (1989): Provided a consistent specification, ensuring cross-platform compatibility.
  - ISO C (1990+): Periodic updates like C99, C11, and C18 introduced modern features such as multithreading and performance optimizations.

  Importance:

  - Foundation for Modern Languages: Influenced languages like C++, Java, and Python.
  - System-Level Programming: Essential for operating systems, embedded systems, and drivers.
  - Portability and Performance: Runs efficiently across hardware platforms.

-C's blend of power and simplicity ensures its continued relevance.

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

- 1. Embedded systems: It is used in microcontrollers, IoT devices, automotive systems and medical equipment's and systems. C is lightweight, provides low-level hardware access, and has a small runtime footprint, making it ideal for resource-constrained environments.

- 2. Operating Systems: It is used in developing system kernels and system-level software's such as Linux, Windows, macOS and mobile OS kernels like Android. C allows direct manipulation of hardware resources like memory and CPU registers, providing the control needed for low-level system operations.

- 3. Operating Systems: It is used for game engines and performance critical game components. C offers high performance, memory management capabilities, and close-to-hardware operations, which are essential for rendering graphics and handling real-time inputs.

Q-2 TE: Describe the steps to install a C compiler (e.g., GCC) and set up an Integrated Development Environment (IDE) like DevC++, VS Code, or Code Blocks.

- Download a C compiler exe file, then Install on your system and configure the IDE.

LE:- Write your first program to print "Hello, World!" and run it.

```
#include<stdio.h>

Main()

{

    printf("Hello, World!");

 }
```

Output: Hello, World!

Q-3 TE: Explain the basic structure of a C program, including headers, main function, comments, data types, and variables. Provide examples.

1. Headers: Headers include preprocessor directives, which provide functionality like input/output operations or mathematical computations.
   Ex.: #include <stdio.h>
2. Main Function: Every C program must have a main() function, which serves as the entry point.
   Ex.: int main()
   {
   // Code execution starts here
   return 0;
   }
3. Comments: Comments are used to explain code and are ignored by the compiler. Single-line comments start with //, and multi-line comments are enclosed within /* ... */.

Ex.: // This is a single-line comment
/* This is a multi-line comment */

4. Data Types: Data types define the kind of data a variable can hold, such as integers, floating-point numbers, or characters. Common data types include int, char, double, and float.
   Ex.: int = 1;
   float = 1.2;
   char = 'J';
5. Variables: Variables store data that can be used and modified during program execution. Variables must be declared with a type before use.
   Ex.: int a = 1;
   int b = 2;
   int sum = a + b;

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

```c
#include <stdio.h>

main()
{

    int age = 22;
    char initial = 'J';
    float rs = 10.2;

    const int MAX_VALUE = 50;

    printf("Age: %d\n", age);
    printf("Initial: %c\n", initial);
    printf("RS: %f\n", rs);

    printf("Maximum allow value: %d\n", MAX_VALUE);
}
```

```
Age: 22
Initial: J
RS: 10.200000
Maximum allow value: 50
```

Q-4 TE: Write notes explaining each type of operator in C: arithmetic, relational, logical, assignment, increment/decrement, bitwise, and conditional operators.

- Arithmetic Operators: For mathematical operation
  - + Addition
  - – Subtraction
  - * Multiplication
  - / Division
  - % Modulus

- Relational Operators: For Compare two values and return true or false
  - == Equal to
  - != Not Equal to
  - < Less then
  - > Greater then
  - <= Less then equal to
  - >= Greater then equal to
- Logical Operators: For logical expression
  - && AND
  - || OR
  - | NOT
- Assignment Operators: for modify value of variables
  - = Assign value
  - += Add and Assign
  - -= Subtract and Assign
  - *= Multiply and Assign
  - /= Divide and Assign
  - %= Modulus and Assign
- Increment/Decrement Operators: For increase or decrease value
  - ++ Increment operator
  - -- Decrement operator
- Bitwise Operators: Operate at bit level
  - & Bitwise And operator
  - | Bitwise Or operator
  - ^ Bitwise XOR operator
  - ~ Bitwise NOT operator
  - << Bitwise left shift operator
  - >> Bitwise right shift operator
- Conditional Operators: Simplifies if-else statement
  - Syntax: *variable = (condition) ? Expression2 : Expression3*;

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

```c
#include <stdio.h>

main()
{

    int num1, num2;

    printf("Enter the first integer: ");
    scanf("%d", &num1);
    printf("Enter the second integer: ");
    scanf("%d", &num2);

    // Arithmetic operation
    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);

    // Relational operation
    printf("%d == %d : %d\n", num1, num2, num1 == num2);
    printf("%d != %d : %d\n", num1, num2, num1 != num2);
    printf("%d > %d : %d\n", num1, num2, num1 > num2);
    printf("%d < %d : %d\n", num1, num2, num1 < num2);
    printf("%d >= %d : %d\n", num1, num2, num1 >= num2);
    printf("%d <= %d : %d\n", num1, num2, num1 <= num2);

    // Logical operation
    printf("(%d > 0) && (%d > 0): %d\n", num1, num2, (num1 > 0) && (num2 > 0));
    printf("(%d > 0) || (%d > 0): %d\n", num1, num2, (num1 > 0) || (num2 > 0));
    printf("!(%d > %d): %d\n", num1, num2, !(num1 > num2));
}
```

```
Enter the first integer: 4
Enter the second integer: 5
Addition: 4 + 5 = 9
Subtraction: 4 - 5 = -1
Multiplication: 4 * 5 = 20
4 == 5 : 0
4 != 5 : 1
4 > 5 : 0
4 < 5 : 1
4 >= 5 : 0
4 <= 5 : 1
(4 > 0) && (5 > 0): 1
(4 > 0) || (5 > 0): 1
!(4 > 5): 1
```

Q-5 TE: Explain decision-making statements in C (if, else, nested if-else, switch). Provide examples of each.

1. if Statement: if statement checks a condition and executes a code when if the condition is true.
   Ex.: int age = 22;
   ```
   if (age >= 22)
   {
       printf("You are an adult.\n");
   }
   ```
2. else Statement: else statement follows an if statement and is executed if the if condition is false.
   Ex.: int age = 15;
   ```
   if (age >= 18)
   {
       printf("You are an adult.\n");
   }
   else
   {
       printf("You are a minor.\n");
   }
   ```
3. if-else statement: if-else statement allows the program to choose between two blocks of code, based on a condition.
   Ex.: int number = 10;
   ```
   if (number % 2 == 0)
   {
       printf("The number is even.\n");
   }
   else
   {
       printf("The number is odd.\n");
   }
   ```
4. Nested if-else statement: A nested if-else statement means an if-else structure inside another if-else. It is used when there are multiple conditions to check.
   Ex.: int age = 20;
   ```
   if (age >= 18)
   {
       if (age >= 21)
       {
           printf("You are eligible to drink alcohol.\n");
       }
       else
       {
   ```

```
            printf("You are an adult but not eligible to drink alcohol.\n");
        }
    }
    else
    {
        printf("You are a minor.\n");
    }
```

5. switch Statement: switch statement is used to select one of many code blocks to be executed based on the value of a variable.

   Ex.:
```
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");
}
```

LE.: 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.).

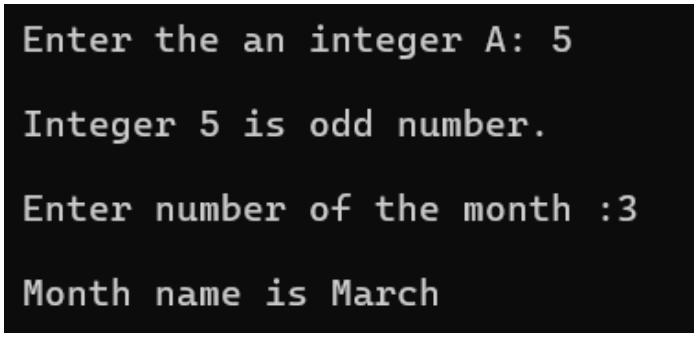
```c
#include <stdio.h>

main()
{

    int num, mon;
    printf("Enter the an integer A: ");
    scanf("%d", &num);

    if(num%2==0)
    {
        printf("\nInteger %d is even number.", num);
    }
    else
    {
        printf("\nInteger %d is odd number.", num);
    }
    printf("\n");
    printf("\nEnter number of the month :");
    scanf("%d", &mon);

    // Displaying month name
    switch(mon)
    {
     case 1:
         printf("\nMonth name is January");
         break;
     case 2:
         printf("\nMonth name is February");
         break;
     case 3:
         printf("\nMonth name is March");
         break;
     case 4:
         printf("\nMonth name is April");
         break;
     case 5:
         printf("\nMonth name is May");
         break;
     case 6:
         printf("\nMonth name is June");
         break;
     case 7:
         printf("\nMonth name is July");
         break;
     case 8:
         printf("\nMonth name is August");
         break;
     case 9:
         printf("\nMonth name is September");
         break;
     case 10:
         printf("\nMonth name is October");
         break;
     case 11:
         printf("\nMonth name is November");
         break;
     case 12:
         printf("\nMonth name is December");
         break;
     default:
         printf("\nMonth number is invalid");
    }
}
```



Q-6 TE: Compare and contrast while loops, for loops, and do-while loops. Explain the scenarios in which each loop is most appropriate.

- while loop: it is a pre-checking loop so the condition is evaluated before executing the block of code. Thus, it is used were it required to verify the condition before executing a block of code. The code may not execute if the condition is false initially.
    - Reading input until the user enters a specific value or processing data until a file ends.
- for loop: It is a count-controlled loop, making it ideal when the number of iterations are known beforehand. It combines the initialisation, condition-checking and increment/decrement in a single line. It is best for situations with a fixed number of iterations.
    - Iterating over arrays, counting numbers, or running a loop a specific number of times.
- Do-while loop: This loop is a post-checking loop, i.e., the condition is evaluated after executing the block of code. This loop is used where it is necessary to execute the block of code at least once. It is also used where the code is executed first and condition is checked afterwards.
    - Menu-driven programs or input validation where initial execution is required.

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

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

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

    // for loop
    for(i=0;i<11;i++)
    {
        if(i!=0)
        {
            printf("%d ", i);
        }
    }
    printf("\n");
    // do..while loop
    b=1;
    do
    {
        printf("%d ", b);
        b++;
    }
    while(b<=10);
}
```

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

Q-7 TE: Explain the use of break, continue, and goto statements in C. Provide examples of each.

1. break Statement: The break statement is used to exit from a loop or switch statement prematurely.
   Ex.:
```
#include <stdio.h>
int main()
{
 for (int i = 1; i <= 5; i++)
 {
       if (i == 3)
       {
         break;   // Exit the loop when i equals 3
       }
       printf("%d ", i);
 }
}
```
2. Continue Statement: Continue statement is used to skip the rest of the code in the current iteration of a loop and move to the next iteration.
   Ex.:
```
#include <stdio.h>
int main()
{
       for (int i = 1; i <= 5; i++)
       {
             if (i == 3)
             {
                   continue; // Skip the iteration when i equals 3
              }
        printf("%d ", i);
        }
}
```
3. Go-to Statement: Go-to statement transfers control to a labeled statement elsewhere in the code.
   Ex.:
```
#include <stdio.h>
int main()
{
     for (int i = 1; i <= 3; i++)
     {
           for (int j = 1; j <= 3; j++)
          {
               if (i * j == 4)
              {
                    goto exit_loops; // Exit both loops
               }
```

```
                printf("%d ", i * j);
            }
        }

    exit_loops:
    printf("\nExited loops.\n");

}
```

LE.: 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.

```c
#include <stdio.h>
main()
{
    int a=1;
    while(a<=10)
    {
        if(a==2)
        {
            a++;
            continue;
        }
        else
        {
            printf("%d ", a);
            a++;
        }
    }
        printf("\n");
}
```

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

Q-8 TE: What are functions in C? Explain function declaration, definition, and how to call a function. Provide examples.

- A function is a self-contained block of code that performs a specific task.
- It helps to break down a program into smaller, manageable modules, making the code more organized, reusable, and easier to understand.

- Declaration: In a function declaration, we must provide the function name, its return type, and the number and type of its parameters.
  Ex.: int sum(int *a*, int *b*); //Declaration
- Definition: The function definition consists of actual statements which are executed when the function is called
  Ex.: function_name (para1_type para1_name)
  {
  // body of the function
  }
- Call function: A function call is a statement that instructs the compiler to execute the function.

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

```c
#include <stdio.h>
factorial(int n)
{
    int fac=1;
    int i;

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

main()
{
    int a;
    printf("Enter a number : ");
    scanf("%d", &a);
    printf("\nFactorial value is %d", factorial(a));
}
```

```
Enter a number : 4

Factorial value is 24
--------------------------------
```

Q-9 TE: Explain the concept of arrays in C. Differentiate between one-dimensional and multi-dimensional arrays with examples.

- In C, an array is a collection of values of the same type that are stored in a variable. One-dimensional arrays store elements in a single line, while multi-dimensional arrays store elements in a table format
- Arrays are fundamental structures that help make data management more efficient and organized.
- There are no limits on the number of dimensions in an array, but specific implementations may have limits.
- One-dimensional arrays:
  - Store elements in a single line
  - Also known as vectors
  - Can be accessed using a pointer, an unsized array, or a sized array
  - One-dimensional array: int arr[5] = {1, 2, 3, 4, 5}
- Multi-dimensional arrays:
  - Store elements in a table format
  - Also known as matrices
  - Can have any number of dimensions, such as two-dimensional (2D) or three-dimensional
  - Two-dimensional array: int a[][3]={ {0,2,3}, {2,1,2} }

LE.: 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>
main()
{
    int ar1[5];
    int i;

    printf("Please enter 5 integer values : ");
    for(i=0;i<5;i++)
    {
        scanf("%d", &ar1[i]);
    }

    printf("\nStored integers are :");
    for(i=0;i<5;i++)
    {
        printf("\n%d", ar1[i]);
    }
}
```

```
Please enter 5 integer values : 1
2
3
4
5

Stored integers are :
1
2
3
4
5
```

```c
#include<stdio.h>
main()
{
    int ar1[3][3];
    int sum=0;
    int i,j;

    for(i=0;i<3;i++)
    {
        printf("\nPlease enter 3 integer values : ");
        for(j=0;j<3;j++)
        {
            scanf("%d", &ar1[i][j]);
        }
    }
    printf("\nStored integers are : \n");
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("%d ", ar1[i][j]);
            sum = sum + ar1[i][j];
        }
        printf("\n");
    }
    printf("\nSum of all numbers is %d", sum);
}
```

```
Please enter 3 integer values : 1
2
3

Please enter 3 integer values : 4
5
6

Please enter 3 integer values : 7
8
9

Stored integers are :
1 2 3
4 5 6
7 8 9

Sum of all numbers is 45
```

Q-10 TE: Explain what pointers are in C and how they are declared and initialized. Why are pointers important in C?

- A pointer is a variable that holds the memory address of another variable. Pointers are essential for efficient memory management and data manipulation.
- Pointers declared and initialized in C:
  - To declare a pointer, use the syntax type *var-name
  - To initialize a pointer, use the & operator to get the address of a variable and assign it to the pointer
  - To assign a null value to a pointer, use the NULL macro or assign it the value 0
- Pointers important in C:
  - Pointers allow for direct memory access, which enables programmers to work with complex data structures.
  - Pointers allow for dynamic memory allocation, which enables the return of multiple values from functions.
  - Pointers can reduce code and improve performance.

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

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

    int num = 21;
    int *ptr = &num;

    printf("Value of num: %d\n", num);
    printf("Address of num: %p\n", &num);
    *ptr = 39;
    printf("Value of num after : %d\n", num);
}
```

```
Value of num: 21
Address of num: 000000000062FE44
Value of num after : 39
```

Q-11 TE: Explain string handling functions like strlen(), strcpy(), strcat(), strcmp(), and strchr(). Provide examples of when these functions are useful.

- strlen(str): Returns the length of a string (number of characters) excluding the null terminator
  Ex: char name[] = "Jil";
  int length = strlen(name); // length will be 3 (J,I,l)
- strcpy(dest, src): Copies the entire string pointed to by src into the memory location pointed to by dest, including the null terminator.
  Ex.: char str1[20] = "Hello";
  char str2[20];
  strcpy(str2, str1); // str2 now contains "Hello"
- strcat(dest, src): Concatenates (appends) the string pointed to by src to the end of the string pointed to by dest.
  Ex.: char greeting[] = "Hi";
  char name[] = "John";
  strcat(greeting, name); // greeting becomes "HiJohn"
- strcmp(str1, str2): Compares two strings lexicographically and returns an integer value
  Ex.: char str1[] = "apple";
  char str2[] = "banana";
  int result = strcmp(str1, str2); // result will be a negative value
- strchr(str, ch): Returns a pointer to the first occurrence of the character ch within the string str, or NULL if not found.
  Ex.: char sentence[] = "The quick brown fox";
  char *found = strchr(sentence, 'q'); // found will point to the 'q' character

LE.: 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().

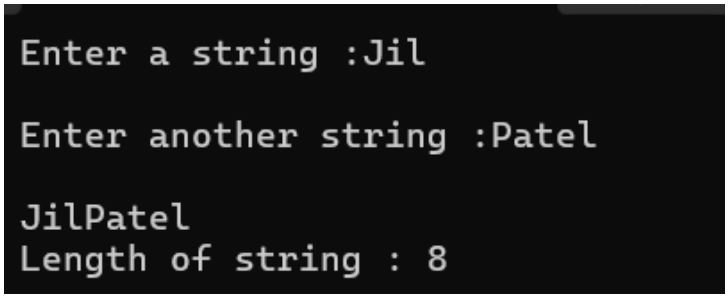
```c
#include <stdio.h>
#include <string.h> // header file for managing strings
main()
{

    // Declare strings
    char st1[20], st2[20], res[50];

    //get strings
    printf("Enter a string :");
    fgets(st1, sizeof(st1), stdin); // Read string
    st1[strcspn(st1, "\n")] = '\0'; // remove newline character

    printf("\nEnter another string :");
    fgets(st2, sizeof(st2), stdin);
    st2[strcspn(st2, "\n")] = '\0';

    strcpy(res, st1); // copy string in new variable
    strcat(res, st2); // Concatenate strings
    printf("\n%s", res); // print string
    printf("\nLength of string : %lu", strlen(res)); // print length of string
}
```



Q-12 TH: Explain the concept of structures in C. Describe how to declare, initialize, and access structure members.

- A structure is a user-defined data type that allows you to group together variables of different data types under a single name. It's a way to represent a collection of related data items.
- Declaring a Structure: To declare a structure, you use the struct keyword followed by the structure name and a set of curly braces containing the member variables.

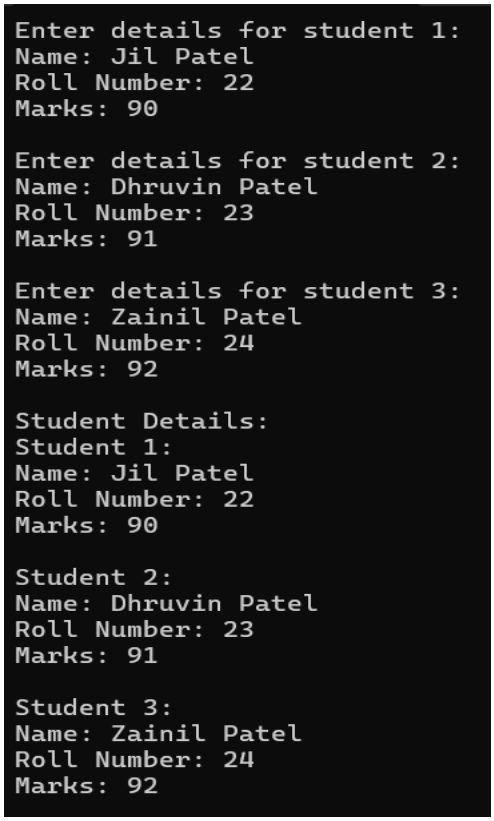
```
struct Employee
{
   int id;
  char name[50];
  float salary;
};
```

- Initializing a Structure: You can initialize a structure in several ways: During declaration.
    - struct Employee emp1 = {1, "John", 5000.0};
- Accessing Structure Members: You can access structure members using the dot (.) operator.

```
printf("Employee ID: %d\n", emp1.id);
printf("Employee Name: %s\n", emp1.name);
printf("Employee Salary: %.2f\n", emp1.salary);
```

LE.: 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 3students and print them.

```c
#include <stdio.h>
struct Student // Define the structure to store student details
{
    char name[50];
    int rollnum;
    int marks;
};
main()
{
    struct Student students[3]; // Array of structures for 3 students
    int i;
    for (i=0;i<3;i++) // Input details of each student
    {
        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].rollnum);
        printf("Marks: ");
        scanf("%d", &students[i].marks);
        printf("\n");
    }
    printf("Student Details:\n");
    for (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].rollnum);
        printf("Marks: %d\n", students[i].marks);
        printf("\n");
    }
}
```



Q-13 TH: Explain the importance of file handling in C. Discuss how to perform file operations like opening, closing, reading, and writing files.

- File handling in C is crucial because it allows programs to interact with persistent data stored on the computer's hard drive, enabling them to read information from existing files, write new data to files, and modify existing data, which is essential for creating applications that need to store and retrieve data beyond the program's execution lifespan.
- Opening a file: fopen(filename, mode)
    - This function takes the file name as a string and a mode string (like "r" for read, "w" for write, "a" for append) and returns a file pointer, which is a special variable used to reference the opened file
    - Ex.: FILE *myFile;
      myFile = fopen("data.txt", "r");
- Closing a file: fclose(filePointer)
    - This function is used to release the file handle and properly close the file, which is important to avoid data corruption and ensure the file is saved correctly.
    - Ex.: fclose(myFile);
- Reading file: fgetc(filePointer) (reads a single character), fscanf(filePointer, format, ...) (reads formatted data)

- These functions read data from the file pointed to by the file pointer according to the specified format.
- Ex.: char ch;
  while ((ch = fgetc(myFile)) != EOF)
  {
  printf("%c", ch);
  }

- Writing file: fputc(character, filePointer) (writes a single character), fprintf(filePointer, format, ...) (writes formatted data)
  - These functions write data to the file pointed to by the file pointer.
  - Ex.: fprintf(myFile, "This is some data to write to the file.\n");

LE.: 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.

```c
#include <stdio.h>
#include <stdlib.h>
main()
{
    FILE *file;
    char filename[100] = "example.txt"; // File name
    char content[100];

    file = fopen(filename, "w"); // Open the file in write mode
    if (file == NULL)
    {
        printf("Error: Unable to create file.\n");
        exit(1);
    }
    printf("Enter a string to write to the file: ");
    fgets(content, sizeof(content), stdin); // Read a string from the user

    fprintf(file, "%s", content); // Write the string to the file
    fclose(file); // Close the file after writing
    printf("String written to file successfully.\n");

    file = fopen(filename, "r"); // Open the file in read mode
    if (file == NULL)
    {
        printf("Error: Unable to open file.\n");
        exit(1);
    }
    printf("\nContents of the file:\n");
    while (fgets(content, sizeof(content), file) != NULL)
    {
        printf("%s", content); // Read and print the file contents
    }
    fclose(file); // Close the file after reading

}
```

```
Enter a string to write to the file: Jil Patel
String written to file successfully.

Contents of the file:
Jil Patel
```

EXTRA LAB EXERCISES FOR IMPROVING PROGRAMMING LOGIC

❖ Operators

LE 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.

```c
#include<stdio.h>
main()
{
    int nu1, nu2;
    char operation;

    printf("Please enter first number : ");
    scanf("%d", &nu1);

    printf("\nPlease select any one operator(+,-,*,/,%%): ");
    scanf(" %c", &operation);

    printf("\nPlease enter second number : ");
    scanf("%d", &nu2);

    switch(operation)
    {
        case '+':
            printf("Result: %d\n", nu1+nu2);
            break;
        case '-':
            printf("Result: %d\n", nu1-nu2);
            break;
        case '*':
            printf("Result: %d\n", nu1*nu2);
            break;
        case '/':
            if(nu2 != 0){
                printf("Result: %.2f\n", (float)nu1 / nu2);
                break;
            } else {
                printf("Division by zero is not allowed.\n");
                break;
            }
        case '%':
            if(nu2 != 0)
            {
                printf("Result: %d\n", (nu1%nu2));
                break;
            } else
            {
                printf("Division by zero is not allowed.\n");
                break;
            }
        default:
            printf("Selected operator is invalid.\n");
    }
}
```

```
Please enter first number : 5

Please select any one operator(+,-,*,/,%): +

Please enter second number : 6
Result: 11
```

LE 1.: 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.

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

    if (a % 2 == 0) // Check if the number is even
    {
        printf("The number is even\n");
    }
    else
    {
        printf("The number is odd\n");
    }

    if (a > 0) // Check if the number is positive, negative or zero
    {
        printf("The number is positive\n");
    }
    else if (a < 0)
    {
        printf("The number is negative\n");
    }
    else
    {
        printf("The number is zero\n");
    }

    if (a%3 == 0 && a%5 == 0) // Check if the number is divisible by 3 and 5
    {
        printf("The number is divisible by 3 and 5\n");
    }
    else if (a%3 == 0)
    {
        printf("The number is divisible by 3\n");
    }
    else if (a%5 == 0)
    {
        printf("The number is divisible by 5\n");
    }
    else
    {
        printf("The number is not divisible by 3 or 5\n");
    }
}
```

```
Please enter a number: 4
6
The number is even
The number is positive
The number is not divisible by 3 or 5
```

2. Control Statements

LE-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

```c
#include<stdio.h>
main()
{
    int marks;
    printf("Please enter student's marks :");
    scanf("%d", &marks);

    if(marks >= 50) // Check if the marks are greater than 50
    {
        if(marks > 75) // Check if the marks are greater than 75
        {
            if(marks > 90) // Check if the marks are greater than 90
            {
                printf("Grade A\n");
            }
            else
            {
                printf("Grade B\n");
            }
        }
        else
        {
            printf("Grade C\n");
        }
    }
    else
    {
        printf("Grade D\n");
    }
}
```

```
Please enter student's marks :90
Grade B
```

```
Please enter student's marks :55
Grade C
```

LE 2: Number Comparison:
Write a C program that takes three numbers from the user and determines:
o The largest number.
o The smallest number.

```c
#include<stdio.h>
main()
{
    int a, b, c;  //With If-Else

    printf("Enter the value of first number: ");
    scanf("%d", &a);
    printf("Enter the value of second number: ");
    scanf("%d", &b);
    printf("Enter the value of third number: ");
    scanf("%d", &c);

    if(a >= b){ // a>=b
        if(a >= c){  // a>=b and a>=c
            if(c >= b){  // a>=b and a>=c and c>=b
                printf("Largest number is: %d\n", a);
                printf("Smallest number is: %d\n", b);
            }
            else{ // a>=b and a>=c and b>=c
                printf("Largest number is: %d\n", a);
                printf("Smallest number is: %d\n", c);
            }
        }
        else{ // a>=b and c>=a
            printf("Largest number is: %d\n", c);
            printf("Smallest number is: %d\n", b);
        }
    }
    else{
        if(b >= c){
            if(c >= a){ // b>=a and b>=c and c>=a
                printf("Largest number is: %d\n", b);
                printf("Smallest number is: %d\n", a);
            }
            else{ // b>=a and b>=c and a>=c
                printf("Largest number is: %d\n", b);
                printf("Smallest number is: %d\n", c);
            }
        }
        else{
            if(a >= b){ // c>=b and c>=a and a>=b
                printf("Largest number is: %d\n", c);
                printf("Smallest number is: %d\n", b);
            }
            else{ // c>=b and c>=a and b>=a
                printf("Largest number is: %d\n", c);
                printf("Smallest number is: %d\n", a);
            }
        }
    }
}
```

```c
#include<stdio.h>
main()
{
    int a, b, c; //Switch case

    printf("Enter the value of first number: ");
    scanf("%d", &a);
    printf("Enter the value of second number: ");
    scanf("%d", &b);
    printf("Enter the value of third number: ");
    scanf("%d", &c);

    switch(a >= b)
    {
        case 1:
        switch(a >= c)
        {
            case 1:
                printf("Largest number is %d\n", a);
                break;
            case 0:
                printf("Largest number is %d\n", c);
                break;
        }
        break;
        case 0:
        switch(b >= c)
        {
            case 1:
                printf("Largest number is %d\n", b);
                break;
            case 0:
                printf("Largest number is %d\n", c);
                break;
        }
        break;
    }

    // Calculate smallest number
    switch(a<=b){
        case 1:
            switch(a<=c)
            {
                case 1:
                    printf("Smallest number is %d\n", a);
                    break;
                case 0:
                    printf("Smallest number is %d\n", c);
                    break;
            }
            break;
        case 0:
            switch(b<=c)
            {
                case 1:
                    printf("Smallest number is %d\n", b);
                    break;
                case 0:
                    printf("Smallest number is %d\n", c);
                    break;
            }
            break;
```

```
Enter the value of first number: 5
Enter the value of second number: 6
Enter the value of third number: 7
Largest number is: 7
Smallest number is: 5
```

```
Enter the value of first number: 1
Enter the value of second number: 2
Enter the value of third number: 3
Largest number is 3
Smallest number is 1
```

3.Loops

LE-1: Prime Number Check

Write a C program that checks whether a given number is a prime number or not using a for loop.

```c
#include<stdio.h>
bool isPrime(int n) // Check if number is prime number
{
    int i;
    if (n <= 1)
    {
        return false; // 0 and 1 are not prime numbers
    }
    for (i=2;i*i<=n; i++)
    {
        if (n % i == 0)
        {
            return false; // not a prime number
        }
    }
    return true; // a prime number
}
main()
{
    int myNumber;
    int j=0;
    int i;

    printf("Please enter a number : ");
    scanf("%d", &myNumber);

    if(isPrime(myNumber))
    {
        printf("%d is a prime number.\n", myNumber);
    }
    else
    {
        printf("%d is not a prime number.\n", myNumber);
    }

    printf("Prime numbers between 1 and %d :\n", myNumber);
    for (i=2; i<= myNumber; i++)
    {
        if (isPrime(i))
        {
            printf("%d, ", i);
            j++;
            if(j%10==0)
            {
                printf("\n");
            }
        }
    }
}
```

```
Please enter a number : 5
5 is a prime number.
Prime numbers between 1 and 5 :
2, 3, 5,
```

LE-2: Multiplication Table

Write a C program that takes an integer input from the user and prints its multiplication table using a for loop.

```c
#include <stdio.h>
main()
{
    printf("Multiplication table creator:\n");
    int n, range;
    int i;

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

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

```
Multiplication table creator:
Enter the number: 5
Enter the range: 5
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
```

LE-3: Sum of Digits

Write a C program that takes an integer from the user and calculates the sumof its digits using a while loop.

```c
#include <stdio.h>
main()
{
    int num, sum = 0, remainder;
    printf("Enter an integer: ");
    scanf("%d", &num);
    int temp = num;
    while (temp != 0)
    {
        remainder = temp % 10;  // 4 - 3 - 2 - 1
        sum = sum + remainder;  // 4 - 7 - 9 - 10
        temp = temp / 10;       // 123 - 12 - 1 - 0
    }

    printf("Sum of digits: %d\n", sum);

    int reversed = 0;   // Reverse the digits
    temp = num;
    while (temp != 0)
    {
        remainder = temp % 10; // 4 - 3 - 2 - 1
        reversed = reversed * 10 + remainder; // 4 - 43 - 432 - 4321
        temp = temp / 10; // 123 - 12 - 1 - 0
    }
    printf("Reversed number: %d\n", reversed);
}
```

```
Enter an integer: 56
Sum of digits: 11
Reversed number: 65
```

4. Arrays

LE-1: Maximum and Minimum in Array

Write a C program that accepts 10 integers from the user and stores the min an array. The program should then find and print the maximum and minimum values in the array.

```c
#include <stdio.h>
main()
{
    int arr[10];
    int i,j, max, min, temp;
    printf("Enter 10 integers: ");
    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];// find max
        }
        if (arr[i] < min)
        {
            min = arr[i];// find min
        }
    }
    printf("Max: %d\n", max);
    printf("Min: %d\n", min);
    for (i=0;i<10;i++)// Sort the array
    {
        for (j=i+1;j<10;j++)
        {
            if (arr[i] > arr[j])
            {
                temp = arr[i];// swap elements
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
    printf("Sorted array: ");
    for (i=0;i<10;i++)
    {
        printf("%d, ", arr[i]);
    }
}
```

```
Enter 10 integers: 1
2
3
4
5
6
7
8
9
10
Max: 10
Min: 1
Sorted array: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
```

LE-2: Matrix Addition

Write a C program that accepts two 2x2 matrices from the user and adds them. Display the resultant matrix.

```c
#include <stdio.h>
main()
{
    int a[2][2], b[2][2], c[2][2], d[2][2];
    int i, j;

    printf("Enter the elements of the first 2x2 matrix: \n");
    for (i = 0; i < 2; i++)
    {
        for (j = 0; j < 2; j++)
        {
            scanf("%d", &a[i][j]);
        }
    }
    printf("The first matrix is: \n");// print the first matrix
    for (i = 0; i < 2; i++)
    {
        for (j = 0; j < 2; j++)
        {
            printf("%d ", a[i][j]);
        }
        printf("\n");
    }
    printf("Enter the elements of the second 2x2 matrix: \n");
    for (i = 0; i < 2; i++)
    {
        for (j = 0; j < 2; j++)
        {
            scanf("%d", &b[i][j]);
        }
    }
    printf("The second matrix is: \n");// print the second matrix
    for (i = 0; i < 2; i++) {
        for (j = 0; j < 2; j++)
        {
            printf("%d ", b[i][j]);
        }
        printf("\n");
    }

    for (i = 0; i < 2; i++) // add the two matrices
    {
        for (j = 0; j < 2; j++)
        {
            c[i][j] = a[i][j] + b[i][j];
        }
    }
    printf("The resultant matrix is: \n");// print the resultant matrix
    for (i = 0; i < 2; i++)
    {
        for (j = 0; j < 2; j++)
        {
            printf("%d ", c[i][j]);
        }
        printf("\n");
    }
    d[0][0] = a[0][0]*b[0][0] + a[0][1]*b[1][0];    // matrix multiplication
    d[0][1] = a[0][0]*b[0][1] + a[0][1]*b[1][1];
    d[1][0] = a[1][0]*b[0][0] + a[1][1]*b[1][0];
    d[1][1] = a[1][0]*b[0][1] + a[1][1]*b[1][1];

    printf("The resultant matrix after multiplication is: \n");
    for (i = 0; i < 2; i++)
    {
        for (j = 0; j < 2; j++)
        {
            printf("%d ", d[i][j]);
        }
        printf("\n");
    }
}
```

```
Enter the elements of the first 2x2 matrix:
1
2
3
4
The first matrix is:
1 2
3 4
Enter the elements of the second 2x2 matrix:
5
6
7
8
The second matrix is:
5 6
7 8
The resultant matrix is:
6 8
10 12
The resultant matrix after multiplication is:
19 22
43 50
```

LE-3: Sum of Array Elements

Write a C program that takes N numbers from the user and stores the min an array. The program should then calculate and display the sum of all array elements.

```c
#include <stdio.h>
main()
{
    int arr[10];
    int i,j, max, min, temp;
    printf("Enter 10 integers: ");
    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];// find max
        }
        if (arr[i] < min)
        {
            min = arr[i];// find min
        }
    }
    printf("Max: %d\n", max);
    printf("Min: %d\n", min);
    for (i=0;i<10;i++)// Sort the array
    {
        for (j=i+1;j<10;j++)
        {
            if (arr[i] > arr[j])
            {
                temp = arr[i];// swap elements
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
    printf("Sorted array: ");
    for (i=0;i<10;i++)
    {
        printf("%d, ", arr[i]);
    }
}
```

```
Enter the number of elements: 4
Enter number 1: 1
Enter number 2: 2
Enter number 3: 3
Enter number 4: 4
Array: 1 2 3 4
Sum: 10
Average: 2.50
```

5. Functions

LE-1: Fibonacci Sequence

Write a C program that generates the Fibonacci sequence up to N terms using a recursive function.

```c
#include <stdio.h>
fib(int n)
{
    if (n <= 1)
    {
        return n;
    }
    return fib(n - 1) + fib(n - 2);
}
main()
{
    int n, i;
    printf("Enter the number of terms: ");
    scanf("%d", &n);
    printf("Fibonacci series: ");

    for (i = 0; i < n; i++) {
        printf("%d ", fib(i));
    }
    printf("\n");


    int a = 0, b = 1, c;// iterative method
    for (i = 0; i < n; i++)
    {
        if (i <= 1)
        {
            c = i;
        }
        else
        {
            c = a + b;
            a = b;
            b = c;
        }
    }
    printf("Nth Fibonacci number using iterative method: %d\n", c);
    printf("Nth Fibonacci number using recursive method: %d\n", fib(n - 1));

}
```

```
Enter the number of terms: 3
Fibonacci series: 0 1 1
Nth Fibonacci number using iterative method: 1
Nth Fibonacci number using recursive method: 1
```

LE-2: Factorial Calculation

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

```c
#include <stdio.h>
factorial(int n)
{
    int result = 1;
    int i;
    for (i=1;i<=n;i++)
    {
        result *= i;
    }
    return result;
}

factorial_recursive(int n)// Recursive version
{
    if (n == 0)
    {
        return 1;
    }
    return n * factorial_recursive(n - 1);
}

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

    printf("Factorial of %d is %d\n", n, factorial(n));
    printf("Factorial of %d is %d\n", n, factorial_recursive(n));
}
```

```
Enter a number: 5
Factorial of 5 is 120
Factorial of 5 is 120
```

LE-3: Palindrome Check

Write a C program that takes a number as input and checks whether it is a palindrome using a function.

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

    // reversed integer is stored in reversedNum
    while (num != 0)
    {
        remainder = num % 10;
        reversedNum = reversedNum * 10 + remainder;
        num /= 10;
    }
    // palindrome if orignalNum and reversedNum are equal
    if (originalNum == reversedNum)
        printf("%d is a palindrome.\n", originalNum);
    else
        printf("%d is not a palindrome.\n", originalNum);

    // Challenge: Modify the program to check if a given string is a palindrome.
    char str[100];
    int i, length;
    int flag = 0;
    printf("Enter a string: ");
    scanf("%s", str);

    // Find the length of the string
    for (length = 0; str[length] != '\0'; length++);

    // Check if the string is palindrome or not
    for (i = 0; i < length; i++)
    {
        if (str[i] != str[length - i - 1])
        {
            flag = 1;
            break;
        }
    }

    if (flag)
        printf("%s is not a palindrome.\n", str);
    else
        printf("%s is a palindrome.\n", str);

}
```

```
Enter an integer: 2
2 is a palindrome.
Enter a string: JIL
JIL is not a palindrome.
```

6. Strings

LE-1: String Reversal

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

```c
#include <stdio.h>
stringLength(char str[])
{
    int length = 0;
    while (str[length] != '\0')
    {
        length++;
    }
    return length;
}

reverseString(char str[])
{
    int length = stringLength(str);
    int start = 0;
    int end = length - 1;

    while (start < end)
    {
        char temp = str[start];
        str[start] = str[end];
        str[end] = temp;

        start++;
        end--;
    }
}

main()
{
    char inputString[100];
    int i;
    printf("Enter a string: ");
    fgets(inputString, 100, stdin);

    for (i=0;i<100;i++)
    {
        if (inputString[i] == '\n')
        {
            inputString[i] = '\0';
            break;
        }
    }

    reverseString(inputString);
    printf("Reversed string: %s\n", inputString);
}
```

```
Enter a string: JIL PATEL
Reversed string: LETAP LIJ
```

LE – 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.

```c
#include <stdio.h>
isVowel(char ch) // check if a character is a vowel
{
    char lower = (ch >= 'A' && ch <= 'Z') ? (ch + 32) : ch;
    return (lower == 'a' || lower == 'e' || lower == 'i' || lower == 'o' || lower == 'u');
}

isConsonant(char ch) //check if a character is a consonant
{
    char lower = (ch >= 'A' && ch <= 'Z') ? (ch + 32) : ch;
    return ((lower >= 'a' && lower <= 'z') && !isVowel(ch));
}

isDigit(char ch) // check if a character is a digit
{
    return (ch >= '0' && ch <= '9');
}

isSpecialCharacter(char ch) // check if a character is a special character
{
    return (!isVowel(ch) && !isConsonant(ch) && !isDigit(ch) && ch != ' ' && ch != '\n');
}

main()
{
    char inputString[100];
    int vowels = 0, consonants = 0, digits = 0, specialCharacters = 0;
    int i;
    printf("Enter a string: ");
    fgets(inputString, 100, stdin);
    for (i=0;i<100;i++)  // Remove newline character
    {
        if (inputString[i] == '\n')
        {
            inputString[i] = '\0';
            break;
        }
    }

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

        char ch = inputString[i];
        if (isVowel(ch))
        {
            vowels++;
        } else if (isConsonant(ch))
        {
            consonants++;
        } else if (isDigit(ch))
        {
            digits++;
        } else if (isSpecialCharacter(ch))
        {
            specialCharacters++;
        }
    }

    printf("Vowels: %d\n", vowels);
    printf("Consonants: %d\n", consonants);
    printf("Digits: %d\n", digits);
    printf("Special Characters: %d\n", specialCharacters);

}
```

```
Enter a string: PATEL JIL 24 @
Vowels: 3
Consonants: 5
Digits: 2
Special Characters: 1
```

LE - 3: Word Count

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

```c
#include <stdio.h>
#include <string.h>
#define MAX 100

wordCount(char str[]) // Function to count the number of words
{
    int count = 0;
    int i = 0;

    while (str[i] != '\0')
    {
        if ((str[i] != ' ' && str[i] != '\n') &&
            (str[i + 1] == ' ' || str[i + 1] == '\n' || str[i + 1] == '\0'))
        {
            count++;
        }
        i++;
    }
    return count;
}

findLongestWord(char str[], char longestWord[])
{
    int maxLen = 0;
    int len = 0;
    int start = 0;
    int i;
    for (i=0; str[i] != '\0'; i++)
    {
        if (str[i] != ' ' && str[i] != '\n')
        {
            len++;
        }
        else
        {
            if (len > maxLen)
            {
                maxLen = len;
                strncpy(longestWord, &str[start], len);
                longestWord[len] = '\0';
            }
            len = 0;
            start = i + 1;
        }
    }

    if (len > maxLen) // Check for the last word in the string
    {
        strncpy(longestWord, &str[start], len);
        longestWord[len] = '\0';
    }
}

main()
{
    char inputString[MAX];
    char longestWord[MAX];

    printf("Enter a sentence: ");
    fgets(inputString, MAX, stdin);

    if (inputString[strlen(inputString) - 1] == '\n') // Remove newline character
    {
        inputString[strlen(inputString) - 1] = '\0';
    }

    int count = wordCount(inputString);
    findLongestWord(inputString, longestWord);

    printf("Number of words: %d\n", count);
    printf("Longest word: %s\n", longestWord);

}
```

```
Enter a sentence: JIL PATEL
Number of words: 2
Longest word: PATEL
```

Extra Logic Building Challenges

LE 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).

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

isArmstrong(int num)
{
    int originalNum, remainder, digits = 0, sum = 0;

    originalNum = num;
    while (originalNum != 0) // Calculate the number of digits
    {
        digits++;
        originalNum /= 10;
    }

    originalNum = num;

    while (originalNum != 0)
    {
        remainder = originalNum % 10;
        sum += pow(remainder, digits);
        originalNum /= 10;
    }


    if (sum == num) // Check if the sum is equal to the original number
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

main()
{
    int num;
    printf("Enter a number to check if it's 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:\n");
     for (i=1;i<=1000;i++)
     {
         if (isArmstrong(i))
         {
             printf("%d\n", i);
         }
     }
}
```

```
Enter a number to check if it's an Armstrong number: 15
15 is not an Armstrong number.
Armstrong numbers between 1 and 1000:
1
2
3
4
5
6
7
8
9
153
370
371
407
```

LE-2: Pascal's Triangle

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

```c
#include <stdio.h>
generatePascalsTriangleLoops(int n)
{
    int i;
    printf("Pascal's Triangle using loops:\n");
    for (int line = 0; line < n; line++)
    {
        int value = 1; // First value in every line is 1
        for (i = 0; i <= line; i++)
        {
            printf("%d ", value);
            value = value * (line - i) / (i + 1);
        }
        printf("\n");
    }
}

pascalValue(int row, int col) // Recursive function to get the value at a given position
{
    if (col == 0 || col == row)
    {
        return 1;
    }
    return pascalValue(row - 1, col - 1) + pascalValue(row - 1, col);
}

generatePascalsTriangleRecursive(int n) // Function to generate Pascal's Triangle using recursion
{
    int i;
    printf("Pascal's Triangle using recursion:\n");
    for (int line = 0; line < n; line++)
    {
        for (i = 0; i <= line; i++)
        {
            printf("%d ", pascalValue(line, i));
        }
        printf("\n");
    }
}

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

    generatePascalsTriangleLoops(n);
    printf("\n");
    generatePascalsTriangleRecursive(n);
}
```

```
Enter the number of rows: 6
Pascal's Triangle using loops:
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1

Pascal's Triangle using recursion:
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
```

LE-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.

```c
#include<stdio.h>
main()
{
    printf("\n*********Welcome to the guess number!!*********");
    int n=35;
    int a=5;

    while(a!=0)
    {
        printf("\n*********Enter number between 1 to 50!!*********");

        int ch;
        printf("\nEnter number between 1 to 50!!: ");
        scanf("%d",&ch);

        if(ch>50)
        {
            printf("\nInvalid Number!!");
            break;
        }
        else if(ch == n)
        {
            printf("You win");
            break;
        }
        else if(ch>n)
        {
            printf("\nEntered number are greatest!!");
        }
        else
        {
            printf("\nEntered number are smallest!!");
        }
    a--;
    }
    printf("You Have Use All Attemps....");
}
```

```
*********Welcome to the guess number!!*********
*********Enter number between 1 to 50!!*********
Enter number between 1 to 50!!: 45

Entered number are greatest!!
*********Enter number between 1 to 50!!*********
Enter number between 1 to 50!!: 12

Entered number are smallest!!
*********Enter number between 1 to 50!!*********
Enter number between 1 to 50!!: 33

Entered number are smallest!!
*********Enter number between 1 to 50!!*********
Enter number between 1 to 50!!: 14

Entered number are smallest!!
*********Enter number between 1 to 50!!*********
Enter number between 1 to 50!!: 35
You win
You Have Use All Attemps....
```