LAB-02 **Issue Date:19 Sept 2025**

The objective of this lab is to:

To understand and apply conditional statements and loops for solving basic programming problems.

ALERT!

1. This is an **individual lab**. You are **strictly NOT** allowed to collaborate with others, share screens, or communicate answers in any form.
2. **Use of AI tools (e.g., ChatGPT, Copilot, etc.) is strictly prohibited.** Any AI-generated content will be treated as academic dishonesty.
3. Anyone caught in act of cheating would be awarded an "F" grade in this Lab.

Task 01: 5 marks

Print Diamond Pattern using Nested Loops

Problem Statement:

Write a program that prints a diamond-shaped pattern of stars (*) for a given number of rows n. The diamond should have a symmetric top and bottom part.

Sample Input

Input: n = 5

Output:

```
    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
    *
```

Task 02: 5 marks

Problem Statement: Decimal to Binary Conversion (both Recursive and loops)

You are required to write a C++ program that converts a given decimal number (base 10) into its equivalent binary number (base 2) using a recursive function.

Key Requirements:

1. Input
   - The program should ask the user to enter a non-negative integer ($n \geq 0$) in decimal form.
   - If the input is negative, display an error message because binary representation of negative numbers is not required here.
2. Recursive Approach
   - Do not use loops (for, while) or built-in functions (bitset, stoi, etc.).
   - Instead, use recursion based on the following idea:
     - Divide the number n by 2.
     - The remainder (n % 2) represents the current binary digit.
     - Recursively process the quotient (n / 2) until it becomes 0.
     - Collect the remainders in reverse order to form the binary number.
3. Output
   - Display the binary representation of the input number n.
   - Special case: if n = 0, output should be 0.

Examples of Input/Output

Case 1: Normal number

Input:
Enter a decimal number: 13
Output:
Binary representation of 13 = 1101
Case 2: Smallest valid input
Input:
Enter a decimal number: 0
Output:
Binary representation of 0 = 0
Case 3: Power of 2 (clean binary with single 1 followed by zeros)
Input:
Enter a decimal number: 16
Output:
Binary representation of 16 = 10000
Case 4: Odd number (ends with 1 in binary)
Input:
Enter a decimal number: 25
Output:
Binary representation of 25 = 11001
Case 5: Large number
Input:
Enter a decimal number: 255
Output:
Binary representation of 255 = 11111111
Case 6: Invalid input (negative number)
Input:
Enter a decimal number: -5

Task 03: 5 marks

Function to Swap variables using call by reference

Problem Statement:
Write a function in C++ to **swap two variables without using a third (temporary) variable**. The function must use **call by reference** to receive the variables as arguments, so that the changes are directly reflected in the main function. The swapping should be done using arithmetic operations (addition and subtraction OR multiplication and division).

- You are not allowed to use an extra variable.
- You must use reference parameters.

Requirements:

1. Use **call by reference** for parameters.
2. Do not use a third variable.
3. Input values can be positive, negative, or zero.

Validations:

- If both numbers are the same, swapping should still work correctly (values remain the same).
- Must handle negative numbers and zero properly.
- Avoid division method if one number is zero (since division by zero is not possible).

Function Header:
void swapNumbers(int &a, int &b);
Example Input/Output Patterns:
Case 1: Normal positive integers

Input: a = 5, b = 10
Output before swapping: a = 5, b = 10
Output after swapping: a = 10, b = 5
Case 2: Including zero
Input: a = 0, b = 25
Output before swapping: a = 0, b = 25
Output after swapping: a = 25, b = 0
Case 3: Negative and positive number
Input: a = -7, b = 3
Output before swapping: a = -7, b = 3
Output after swapping: a = 3, b = -7
Case 4: Both negative numbers
Input: a = -4, b = -9
Output before swapping: a = -4, b = -9
Output after swapping: a = -9, b = -4
Case 5: Both numbers same
Input: a = 6, b = 6
Output before swapping: a = 6, b = 6
Output after swapping: a = 6, b = 6

Task 04: 5 marks

Function to Find sum of numbers having call by reference variable as parameter

Problem Statement:
Write a C++ program that defines a function to calculate the sum of digits of a positive integer using call by reference. The function should take two arguments: the number itself and a reference variable to store the result. The function should repeatedly extract the last digit of the number, add it to the result, and reduce the number until it becomes 0.
Requirements:

1. The function must use **call by reference** to update the result.
2. No built-in functions like to_string are allowed.
3. The input must be a positive integer. If the user enters 0 or a negative number, display an error message.

Validations:

- If the number entered is less than or equal to 0, print:
  "Invalid input. Please enter a positive integer."

Examples of Input and Output:
Input:
Enter a number: 1234
Output:
The sum of digits of 1234 = 10
Input:
Enter a number: 987
Output:
The sum of digits of 987 = 24
Input:
Enter a number: 5
Output:
The sum of digits of 5 = 5
Input:
Enter a number: 0
Output:
Invalid input. Please enter a positive integer.

Input:
Enter a number: -46
Output:
Invalid input. Please enter a positive integer.

Task 05: 5 marks

Function to Find strong number having call by reference variable as parameter

Problem Statement: Strong Number Checker (Factorial of Digits)
A **Strong Number** is a number in which the **sum of the factorials of its digits equals the number itself**.
For example:

- 145 is a strong number because:
  1! + 4! +5! =1+24+120=145
- 123 is **not** a strong number because:
  1! + 2!+ 3!=1+2+6=9≠123

Requirements:

1. Write a function in C++ that checks if a given number is a **Strong Number**.
2. The function must use **call by reference** in its parameter to return the sum of factorials of digits.
3. Do not use built-in factorial functions — write your own factorial logic.
4. Input must be a **positive integer**. If the number is 0 or negative, print an error message.

Function Header Example
void checkStrongNumber(int n, int &sum);

- n → input number
- sum → reference variable that stores the sum of factorials of digits

Logic (Step by Step)

1. Take input n.
2. Extract each digit using % 10.
3. Find factorial of that digit.
4. Add the factorial to sum.
5. Continue until all digits are processed.
6. Compare sum with original number.
   - If equal → Strong Number
   - Else → Not Strong Number

Example Inputs and Outputs
Case 1: Strong number
Input:
145
Output:
145 can be represented as 1! + 4! + 5! = 145
145 is a Strong Number
Case 2: Not a strong number
Input:
123
Output:
123 can be represented as 1! + 2! + 3! = 9
123 is NOT a Strong Number
Case 3: Single digit (always strong for 1 and 2)
Input:
1
Output:
1 can be represented as 1! = 1

1 is a Strong Number
Input:
2
Output:
2 can be represented as 2! = 2
2 is a Strong Number
Input:
5
Output:
5 can be represented as 5! = 120
5 is NOT a Strong Number

Case 4: Invalid input (zero or negative)

Input:
0
Output:
Invalid input. Please enter a positive integer.
Input:
-145
Output:
Invalid input. Please enter a positive integer.

Task 06: 5 marks

Function to Print number as highest power of 2 recursively

Problem Statement:
Write a recursive function in C++ that expresses a given positive integer as a sum of powers of 2. Each number must be broken down into the largest possible power of 2, then the remainder is recursively broken down further until the full sum is obtained.

Requirements:

1. You must use recursion. Loops are not allowed.
2. Do not use arrays, bitwise operators, or built-in functions like pow.
3. The input must be a positive integer greater than 0.
4. If the user enters 0 or a negative number, display an error message and terminate.

Validations:

1. Input should be checked to ensure it is a positive integer.
2. Handle incorrect input gracefully (for example, if n <= 0, display: "Invalid input. Please enter a positive integer.").

Examples of Input and Output:
Input:
Enter a number: 13
Output:
13 can be represented as: 8 + 4 + 1
Input:
Enter a number: 1
Output:
1 can be represented as: 1

Input:
Enter a number: 2
Output:
2 can be represented as: 2
Input:
Enter a number: 3
Output:
3 can be represented as: 2 + 1
Input:
Enter a number: 7
Output:
7 can be represented as: 4 + 2 + 1
Input:
Enter a number: 16
Output:
16 can be represented as: 16
Input:
Enter a number: 31
Output:
31 can be represented as: 16 + 8 + 4 + 2 + 1

Task 07: 10 marks

GCD and LCM Using Functions

Problem Statement:
Write a C++ program that defines two functions to calculate the **Greatest Common Divisor (GCD)** and **Least Common Multiple (LCM)** of two numbers
Euclidean Algorithm (Recursive Logic)
The **Euclidean Algorithm** says:

$$\text{GCD}(a,b) = \begin{cases} a & \text{if } b = 0 \\ \text{GCD}(b, a \mod b) & \text{if } b \neq 0 \end{cases}$$

Step-by-step Recursive Logic

1. **Base Case**
   - If b == 0, then a is the GCD.
     (Because any number divides 0, and the last non-zero remainder is the GCD.)
2. **Recursive Case**
   - Otherwise, call the function again with parameters (b, a % b).
   - This keeps reducing the problem until eventually the remainder becomes 0.

The program should:

1. Ask the user to input two positive integers.
2. Validate that both numbers are **greater than 0** (since GCD/LCM of non-positive numbers is undefined).

3. Use the **Euclidean Algorithm** to calculate GCD.
4. Use the formula to calculate LCM:
   LCM(a,b)=a x b/GCD(a,b)
5. Display both GCD and LCM with proper output format.

Validations:

If either number ≤ 0 → Show an error message: "Please enter positive integers only."

Sample Input/Output:

Input 1:

Enter two numbers: 12 18

Output 1:

GCD of 12 and 18 = 6

LCM of 12 and 18 = 36

Input 2:

Enter two numbers: 25 0

Output 2:

Error: Please enter positive integers only.

Task 08: 5 marks

Function to Print Square matrix of size n × n

Problem Statement:

You are required to generate a **square matrix of size n × n** where numbers form **concentric layers** (squares) that decrease towards the center.

- The **outermost layer** of the matrix contains the number n.
- The next inner layer contains the number n-1.
- This process continues until the **center of the matrix** contains the number 1.

The matrix must be printed as output using nested loops.

Rules and Constraints:

1. Use nested loops to build the matrix.
2. Each layer should have the same number throughout.
3. The numbers decrease by 1 as you move one layer inward.
4. Works for both **odd and even values of n**.

Examples:

Input:

Enter size of the Square Matrix:4

Output:

```
4 4 4 4
4 3 3 4
4 3 3 4
4 4 4 4
```

Input:

Enter size of the Square Matrix:5

Output:

```
5 5 5 5 5
5 4 4 4 5
5 4 3 4 5
5 4 4 4 5
5 5 5 5 5
```

5 4 4 4 4 5
5 5 5 5 5 5