LAB-01 **Issue Date: October 2, 2025**

The objective of this lab is to:

To understand pointer notation and apply them in programs.

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]

Digit Frequency Counter

Problem Statement:

Write a program that counts how many times each digit (0–9) occurs in a given number.
Use pointers to update frequency counts inside an array.

Function Header:

```
void countDigits(int *num, int *freq);
```

***num is an integer variable whose digits are to be counted and *freq is an array of size 10 (for digit 0-9).**

Examples:

Case 1:

Input: Enter a number: 122345

Output:

(freq array will be like f[0]=0, f[1]=1, f[2]=2, f[3]=1, f[4]=1, f[5]=1, f[6]=0, f[7]=0, f[8]=0, f[9]=0)
Digit 0 → 0 times
Digit 1 → 1 time
Digit 2 → 2 times
Digit 3 → 1 time
Digit 4 → 1 time
Digit 5 → 1 time …. Digit 9 → 0 time

Case 2:

Input: 1002003

Output:

(freq array will be like f[0]=3, f[1]=1, f[2]=1, f[3]=1, f[4]=0, f[5]=0, f[6]=0, f[7]=0, f[8]=0, f[9]=0)
Digit 0 → 3 times
Digit 1 → 1 time
Digit 2 → 1 time
Digit 3 → 1 time ….. Digit 9 → 0 time

Task 02: [5 Marks]

Hospital – Patient Room Allocation

Problem Statement

A hospital maintains a list of patient room numbers. For proper record keeping and easier allocation, the hospital staff wants the list of room numbers to be **sorted in ascending order**.

You are required to implement a program that:

1. Stores the room numbers in an integer array.
2. Sorts the array using the **Selection Sort algorithm using pointer notation**.
3. Performs swapping using a **pointer-based swap() function** (call by reference).

Function headers:

```
void swap(int *x, int *y);
void sortArray(int *arr, int size);
```

sortArray function implements Selection Sort **and calls swap() whenever two elements need to be exchanged.**

Example:

Case 1:
Before Sorting: 305 102 410 215 120
After Sorting: 102 120 215 305 410

Case 2:
Before Sorting: 501 301 101 401 201
After Sorting: 101 201 301 401 501

Case 3:
Before Sorting: 110 220 330 440 550
After Sorting: 110 220 330 440 550

Task 03: [10 Marks]

Text Editor – Convert Case

Problem Statement:

A text editor has two modes:

1. **Uppercase Mode** – Converts every character in the text to uppercase.
2. **Lowercase Mode** – Converts every character in the text to lowercase.

You are required to implement two functions using **pointer notation only (no array indexing)**.

Function header:

```
void toUpperCaseString(char *str);
void toLowerCaseString(char *str);
```

Explanation:

- The function toUpperCaseString() should check each character of the string using a pointer, and if it is a lowercase letter ('a'–'z'), convert it to uppercase.
- The function toLowerCaseString() should do the opposite: if the character is uppercase ('A'–'Z'), convert it to lowercase.
- ASCII difference: 'a' (97) – 'A' (65) = 32

Example:

Case 1: Uppercase Mode
Input: "Hello Students"
Output: "HELLO STUDENTS"

Case 2: Lowercase Mode
Input: "C++ POINTERS Are FUN!"
Output: "c++ pointers are fun!"

Case 3: Mixed Test
Input: "Lab Assignment #3"
Output after Uppercase: "LAB ASSIGNMENT #3"
Output after Lowercase: "lab assignment #3"

Task 04: [10 Marks]

Array – Consecutive Sum Finder
Problem Statement:
Given an integer array and a target sum, find **all sequences of consecutive elements** whose sum equals the target.
You are required to implement a program that:

1. Uses pointers to traverse the array.
2. Finds all start and end positions of consecutive subarrays that sum to the target.
3. Prints the sequences and their positions.

Function Header:
void findConsecutiveSums(int *arr, int size, int target);
Validations:

- Array size > 0 and ≤ 50.
- Target can be any integer.

Example:
Case 1:
Input:
Array: 2 3 1 4 2 1
Target Sum: 6
Output:
Sequence 1: 2 3 1 (Positions: 0-2)
Sequence 2: 3 1 2 (Positions: 1-3)
Sequence 3: 4 2 (Positions: 3-4)
Case 2:
Input:
Array: 1 2 3 4
Target Sum: 10
Output:
Sequence 1: 1 2 3 4 (Positions: 0-3)

Task 05: [5 Marks]

Fuel Station – Daily Sales Tracker

Problem Statement:
A fuel station records the number of **liters of fuel sold every hour** in a day inside an array.
At the end of the day, the manager wants to analyze the sales data for decision making.

You are required to implement a program that:

1. Stores the hourly sales (liters sold) in an integer array.
2. Calculates the following using a function with **call by reference (pointers)**:
   - The **total fuel sold in the day**.
   - The **highest sale in an hour**.
   - The **lowest sale in an hour**.

Function Header:

void analyzeSales(int *arr, int size, int *total, int *maxSale, int *minSale);

- *arr → stores hourly sales data (array with pointer notation)
- size → number of hours recorded.
- *total → stores the sum of all sales.
- *maxSale → stores the maximum sale.
- *minSale → stores the minimum sale.

The function should traverse the array using pointers and return results via call by reference.
.

Example:

Case 1:
Input:
50 70 30 90 20 60
Output:
Total Fuel Sold = 320 liters
Highest Sale in an Hour = 90 liters
Lowest Sale in an Hour = 20 liters

Case 2:
Input:
10 15 25 5 20
Output:
Total Fuel Sold = 75 liters
Highest Sale in an Hour = 25 liters
Lowest Sale in an Hour = 5 liters

Task 05: [5 Marks]

Matrix Average of Diagonals

Problem Statement:
Write a function hat computes the **average of all diagonal elements** of a integer matrix of size nxn.You should take the size of matrix from user.

- You must consider both the **main diagonal** and the **secondary diagonal**.
- Your logic for computation for diagonal average should be general (For any size enter by user i.e. 3x3, 4x4, 5x5 ….)

Function Header:

double diagonalAverage(const int X[][N], int n);

Example:

Matrix (3x3):
For Input n=3
1 2 3
4 5 6
7 8 9

- Main diagonal → 1, 5, 9
- Secondary diagonal → 3, 5, 7
- Distinct diagonal elements → 1, 5, 9, 3, 7

Sum = 25
Count = 5
Average = 25 / 5 = 5.0

Matrix (4x4)
For n=4

1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16

- Main diagonal → 1, 6, 11, 16
- Secondary diagonal → 4, 7, 10, 13
- Distinct diagonal elements → 1, 6, 11, 16, 4, 7, 10, 13

Sum = 68
Count = 8
Average = 68 / 8 = 8.5

Task 07: [10 Marks]

Array – Peak and Valley Finder
Problem Statement:
A "peak" in an array is an element that is **greater than both its neighbors**.
A "valley" is an element that is **smaller than both its neighbors**.
You are required to implement a program that:

1. Finds the **total number of peaks** and **valleys** in the array.
2. Uses pointer traversal (no array indexing).
3. Prints the positions (indices) of peaks and valleys.

Function Header:
void findPeaksValleys(int *arr, int size, int *numPeaks, int *numValleys);
Validations:
Array size ≥ 3 and ≤ 50 (at least 3 elements to have neighbors).
Example:
Case 1:
Input:
2 5 3 7 6 4
Output:
Number of Peaks = 2 (Positions: 1, 3)
Number of Valleys = 2 (Positions: 2, 5)
Case 2:
Input:

1 2 3 4 5

Output:

Number of Peaks = 0

Number of Valleys = 0

Task 08: [5 Marks]

Vowel & Consonant Counter

Problem Statement:

A program must analyze an input string and count how many vowels and consonants it has. Ignore digits and special symbols.

Use **pointer traversal** (no array indexing).

Function Header:

void countVC(char *str, int *vowels, int *consonants);

Examples:

Case 1:

Input: "Hello World 123!"

Output:

Vowels = 3

Consonants = 7

Case 2:

Input: "Programming in C++"

Output:

Vowels = 5

Consonants = 11

Case 3:

Input: "AEIOU xyz"

Output:

Vowels = 5

Consonants = 3