Data Structure and Algorithm Lab Manual (Week-02)

Lab Instructor: Ms. Rabeeya Saleem

Session: 2024 (Fall 2025)

Example 2.1 – Factorial

```
int factorial(int n) {
    if (n == 0 || n == 1) return 1;   // Base case
    return n * factorial(n - 1);      // Recursive case
}
int main() {
    cout << "Factorial of 5 = " << factorial(5) << endl;
    return 0;
}
```

Example 2.2 – Fibonacci (Naive)

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

int main() {
    cout << "Fibonacci(6) = " << fibonacci(6) << endl; // Output: 8
    return 0;
}
```

Example 2.3 – Sum of Digits

```
int sumDigits(int n) {
    if (n == 0) return 0;
    return (n % 10) + sumDigits(n / 10);
}

int main() {
    cout << "Sum of digits (1524) = " << sumDigits(1524) << endl; // Output: 12
    return 0;
}
```

Example 2.4 – Find Maximum Element

```
int findMax(int arr[], int n) {
    if (n == 1) return arr[0];
    return max(arr[n - 1], findMax(arr, n - 1));
}

int main() {
    int arr[] = {2, 7, 4, 9, 1};
    cout << "Max element = " << findMax(arr, 5) << endl; // Output: 9
    return 0;
}
```

Example 2.5 – Power Function

```
// Iterative function to compute a^b
long long powerIterative(int a, int b) {
    long long result = 1;
    for (int i = 0; i < b; i++) {
        result *= a;
    }
    return result;
}

int main() {
    int a = 2, b = 5;
    cout << a << "^" << b << " = " << powerIterative(a, b) << endl; // Output: 32
    return 0;
}
```

```
// Recursive function to compute a^b
```

```
long long powerRecursive(int a, int b) {
   if (b == 0) return 1;        // base case
   return a * powerRecursive(a, b - 1);  // recursive case
}

int main() {
   int a = 2, b = 5;
   cout << a << "^" << b << " = " << powerRecursive(a, b) << endl; // Output: 32
   return 0;
}
```

Example 2.6 – 2D Matrix Row & Column Sum

```
#include <iostream>
using namespace std;

int main() {
   int mat[3][3] = {{1,2,3}, {4,5,6}, {7,8,9}};
   int rowSum[3] = {0}, colSum[3] = {0};

   for (int i = 0; i < 3; i++) {
      for (int j = 0; j < 3; j++) {
         rowSum[i] += mat[i][j];
         colSum[j] += mat[i][j];
      }
   }

   cout << "Row-wise sum: ";
   for (int i = 0; i < 3; i++) cout << rowSum[i] << " ";
   cout << "\nColumn-wise sum: ";
   for (int j = 0; j < 3; j++) cout << colSum[j] << " ";
   return 0;
}
```

Example 2.7: Transpose of Matrix

```
int mat[rows][cols] = {       {1, 2, 3},       {4, 5, 6}    };
int trans[cols][rows]; // Transposed matrix will be 3x2
   // Compute transpose
   for (int i = 0; i < rows; i++) {
      for (int j = 0; j < cols; j++) {
         trans[j][i] = mat[i][j];
      }
   }
   // Print original matrix
   cout << "Original Matrix (" << rows << "x" << cols << "):\n";
```

```
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            cout << mat[i][j] << " ";
        }
        cout << endl;
    }
    // Print transpose
    cout << "\nTranspose Matrix (" << cols << "x" << rows << "):\n";
    for (int i = 0; i < cols; i++) {
        for (int j = 0; j < rows; j++) {
            cout << trans[i][j] << " ";
        }
        cout << endl;
    }
```

Example 2.8 – Palindrome Check (Recursive)

```
#include <iostream>
using namespace std;

bool isPalindrome(string s, int l, int r) {
    if (l >= r) return true;
    if (s[l] != s[r]) return false;
    return isPalindrome(s, l + 1, r - 1);
}
int main() {
    string word = "madam";
    cout << (isPalindrome(word, 0, word.size() - 1) ? "Palindrome" : "Not Palindrome")
<< endl;
    return 0;
}
```

Example 2.9 – Count Vowels in String

```
#include <iostream>
using namespace std;

int countVowels(string s, int i = 0) {
    if (i == s.size()) return 0;
    char c = tolower(s[i]);
    bool isVowel = (c=='a'||c=='e'||c=='i'||c=='o'||c=='u');
    return (isVowel ? 1 : 0) + countVowels(s, i + 1);
}
int main() {
    cout << "Vowels in 'Engineering' = " << countVowels("Engineering") << endl; //
Output: 5
    return 0;
}
```

Example 2.10 – Remove Spaces

```
#include <iostream>
using namespace std;

string removeSpaces(string s, int i = 0) {
    if (i == s.size()) return "";
    if (s[i] == ' ') return removeSpaces(s, i + 1);
    return s[i] + removeSpaces(s, i + 1);
}

int main() {
    cout << removeSpaces("Data Structures Lab") << endl; // Output: DataStructuresLab
    return 0;
}
```

Problems:

Look for the index of the given element x inthe given array:
`X = [22,2,1,7,11,13,5,2,9]`
`SearchA(Arr, x) – return array of indices`
`Arr: Array`
`x: element to be searched`
	Input: Enter the number: 2
	Output: `Index: 1,7`

Write a function that takes an array as input, starting and ending index and return the index of minimum element from start to ending index in the array. `Minimum(Arr, starting, ending)– return integer`	For example, you are given the following inputs Array: [3,4,7,8,0,1,23,-2,-5] StartingIndex: 4 EndingIndex: 7 Output: (Return index of minimum element)7
Given a number, the task is to find the sum of its digits using an iterative and recursive method. `SumIterative(number) – returns integer`	Input: 1524 Output: `Sum of digits is:12`
Implement a program in **C++** that compresses a string by replacing consecutive occurrences of the same character with the character followed by the count.	Input: xxxyyyzzzz Output: x3y3z4
Given an array, move all `0`s to the end while maintaining the relative order of non-zero elements.	Input: nums = [0,1,0,3,12] Output: [1,3,12,0,0]
Write a program in C++ that takes a **square matrix** as input and calculates: The sum of **upper triangular elements** (including the main diagonal). The sum of **lower triangular elements** (including the main diagonal).	Matrix A (3×3): 1 2 3 4 5 6 7 8 9 UTM: Sum = 1 + 2 + 3 + 5 + 6 + 9 = **26** LTM: Sum = 1 + 4 + 5 + 7 + 8 + 9 = **34**
Array Operations for Employee Management System In this case study, we will demonstrate the application of basic **Array operations** in a real-world scenario. The task is to design a basic **Employee Management System** for a company, where we will use arrays to store employee data and perform various operations such as **insertion**, **deletion**, and **searching** of employee records. **Objective:** The objective of this case study is to utilize array operations to manage and manipulate employee records efficiently. Specifically, we will perform the following array operations: **Insertion**: Adding new employees to the system. **Deletion**: Removing an employee from the system.	=== Employee Management System Demo === --- Insertion --- Inserted: ID=101, Name=Ali, Dept=HR Inserted: ID=102, Name=Sara, Dept=IT Inserted: ID=103, Name=Hassan, Dept=Finance Employees in the System: ID=101, Name=Ali, Dept=HR ID=102, Name=Sara, Dept=IT ID=103, Name=Hassan, Dept=Finance --- Search by ID --- Employee Found: ID=102, Name=Sara, Dept=IT --- Search by Name ---

Search: Finding an employee based on their ID or name. **Display**: Sort them by ID and then Display the record	Employee Found: ID=103, Name=Hassan, Dept=Finance --- Deletion with ID --- Employee with ID=101 deleted successfully. Employees in the System: ID=102, Name=Sara, Dept=IT ID=103, Name=Hassan, Dept=Finance --- Deletion (Non-existing) --- Employee with ID=200 not found! --- Insertion After Deletion --- Inserted: ID=104, Name=Ayesha, Dept=Marketing Employees in the System: ID=102, Name=Sara, Dept=IT ID=103, Name=Hassan, Dept=Finance ID=104, Name=Ayesha, Dept=Marketing