Data Structures and Algorithms — Lab 10

Topic

Exploring Recursive Techniques in C++: A Practical and Analytical Approach.

Objective

1. Implement basic recursive functions in C++ to solve common computational problems
2. Understand the core principles and structure of recursive function calls, including base cases and recursive steps.
3. Explore real-world programming scenarios where recursion provides elegant and efficient solutions.
4. Develop problem-solving skills through dry runs, trace tables, and complexity analysis of recursive algorithms.

Outcomes

1. Demonstrate the ability to write and debug recursive functions in C++.
2. Identify base and recursive cases in a given problem and implement them effectively.
3. Analyze the time and space complexity of recursive solutions using Big-O notation.
4. Apply recursion to solve real-world problems and compare recursive solutions with iterative approaches.

Content

The following sections will be covered during this lab session:

1. **Introduction to Recursion:**
   Begin with foundational recursion exercises such as calculating factorials, Fibonacci numbers, and solving problems like sum of digits. These examples will help solidify the concept of base and recursive cases.
2. **Recursive Implementation of Linked List Operations:**
   Once the basics are clear, we will move toward applying recursion to linked lists:
   - **Singly Linked List:** Implement core operations like insertion, traversal, and deletion using recursive techniques.
   - **Doubly Linked List:** Extend recursive concepts to handle forward and backward navigation, including recursive insertion and deletion operations.
3. **Time and Space Complexity Analysis:**
   Throughout the session, students will analyse the recursive implementations with respect to their time and space complexity, comparing them with traditional iterative approaches.

Instructor: Afham Nazir

Task Instructions

Students are required to complete the following tasks during lab time.

Create a private repository on your GitHub accounts. The name of the repository should be: **Lab-10-DSA**. All files should be uploaded to this repository; this includes the header and cpp files. Please note that the name of class-related files should be the same as the class name. We can name the main file Task_1.cpp or Task.cpp if there is only 1 task. We recommend you work in .h files for classes only since we must work on templates.

Task 1- Sum of Array elements

Write a C++ program that implements a recursive function to calculate the sum of all elements in an integer array. The function should take the array and its size as input and return the total sum using recursion.
You are not allowed to use loops or any built-in functions for summation. Make sure to define a clear base case and recursive case in your implementation.

Task 2- Palindrome Check

Write a C++ program that implements a recursive function to check whether a given string is a palindrome. The program must use recursion to compare characters from both ends of the string and determine if the entire string reads the same forwards and backwards.
You are not allowed to use loops or built-in functions like reverse(), substr(), or erase(). You may only use basic character indexing like str[i].

Task 3-Recursive Maximum Element

Write a C++ program that uses a recursive function to find and return the maximum element in an integer array. The function should operate without using loops or built-in functions, relying solely on recursion to traverse and compare elements.

Sample Output:

```
int arr[] = {12, 5, 18, 7, 3};
int size = 5;

Maximum element: 18
```

Task 4- Linked List Traversal Recursively

Write a C++ program that defines a singly linked list and uses a recursive function to **print all the elements** of the list.

- You should define the linked list node structure.
- Implement a recursive function that takes the head pointer and prints each node's data until the end of the list is reached.
- Do not use any loops for traversing the list

Task 5

Write a C++ program to implement the following singly linked list operations using **recursive functions only** (no loops allowed):

1. **Insertion:**
   - Insert a node at the beginning of the list recursively.
   - Insert a node at the end of the list recursively.
   - Insert a node at a given position recursively.
2. **Deletion:**
   - Delete a node by value recursively.
   - Delete a node at a given position recursively.
3. **Searching:**
   - Search for a value in the list recursively and return its position (1-based index). Return -1 if not found.
4. **Traversal:**
   - Print all elements of the list recursively.

Requirements:

- Define a singly linked list node structure.
- Each operation must be implemented using recursion without any iterative loops.
- Handle edge cases like empty list, invalid positions, and value not found.
- Print the list after every insertion and deletion operation.

Task 6- Doubly LinkedList using Recursion

You are tasked with implementing various operations on a doubly linked list using **recursion only** (no loops allowed). The operations and requirements are as follows:

(a) Basic Operations

Implement the following functions recursively:

- Insert a node at the beginning of the list.
- Insert a node at the end of the list.
- Insert a node at a specific position (1-based index).
- Delete a node by value.
- Delete a node at a given position.
- Search for a value and return its position; return -1 if not found.
- Print the list in forward and reverse order.

(b) Palindrome Check

Write a recursive function to check if the doubly linked list is a palindrome. The function should:

- Recursively compare the data of nodes from the front and rear simultaneously.
- Return true if the list is palindrome, otherwise false.
- Not use loops or any extra data structures (e.g., arrays, stacks).

Task 7- Magic Number **BONUS-QUESTION**

A wizard challenges you to find a **magic number** hidden in a sequence of numbers. The magic number is defined as the largest number in the sequence that is also equal to the sum of any subset of the remaining numbers in the sequence.

Given an array of integers, write a **recursive C++ function** that:

1. Finds the largest number in the array.
2. Checks recursively if this largest number can be represented as a sum of **any subset** (one or more elements) of the remaining numbers.
3. If yes, return that number (the magic number).
4. If no, remove that number and repeat the process with the next largest number.
5. If no such magic number exists, return -1.

Constraints:

- You **must implement the subset sum check recursively** without loops.
- Use recursion to find and test the largest numbers as well.
- The array can contain positive integers only.

Sample Output:

Input: [2, 3, 5, 8, 13]

- Largest number is 13.
- Can 13 be formed by sum of any subset of [2, 3, 5, 8]?
- Yes, because 5 + 8 = 13.
- So, output: 13

Good Luck! Stay focused, think in steps, and enjoy the logic!