Assignment no. 3

Title: Use of Recursion

Part 1 and Part 2 are mandatory.
Part 3 is bonus. You can expect part 3 type questions in the end semester exam.
Lab copy (handwritten part: additional content -- must include the following): *Indicate the decomposition, composition and base case(s) for all the problems in part 1.*

- Part 1
    1. **Array Sum:** a) Implement a recursive function named "arraySum" that calculates the sum of elements in an integer array. b) Create an integer array with a few elements, and then call the "arraySum" function to calculate and display the sum.
    2. **String Reversal:** a) Implement a recursive function named "reverseString" that reverses a given string. b) Prompt the user to input a string, and then call the "reverseString" function to display the reversed string.
    3. **Palindrome Check:** Write a recursive function to determine if a given string is a palindrome. A palindrome is a word, phrase, number, or other sequence of characters that reads the same forward and backward (ignoring spaces, punctuation, and capitalization).
    4. **Power Function:** Create a recursive function to calculate the value of base raised to the power of exponent, where both base and exponent are positive integers. ***Make sure*** *it handles all the boundary cases.*
    5. **Tower of Hanoi:** Solve the classic Tower of Hanoi problem using recursion. Given three pegs and a stack of n disks of different sizes, move the entire stack from one peg to another with the following rules: Only one disk can be moved at a time, and a larger disk cannot be placed on top of a smaller disk.
    6. **Permutations:** Implement a recursive function to generate all possible permutations of a given string. A permutation of a string is an arrangement of its characters in a different order.

- Part 2
    1. **Calculate the Length:** Create a recursive function to find the length (number of nodes) of a linked list. The function should count the number of nodes by recursively calling itself with the next node until it reaches the end.
    2. **Search for an Element:** Implement a recursive function to search for a specific value in the linked list. The function should traverse the list by calling itself with the next node until it finds the target value or reaches the end.
    3. **Merge Two Sorted Lists:** Write a recursive function to merge two sorted linked lists into a single sorted linked list. The function should compare the values of the nodes in both lists and recursively merge them to create a new sorted list.
    4. **Reverse the Linked List:** Implement a recursive function to reverse the linked list. The function should recursively reverse the rest of the list and adjust the next pointers as it traverses back.
    5. **Find the Middle Node:** Write a recursive function to find the middle node of a linked list. The middle node is the one located at *approximately* the center of the list. You can use two pointers to traverse the list—one moving one step at a time and the other moving two steps at a time.
    6. **Parentheses Balancing:** Implement two mutually recursive functions named "isOpenParenthesis" and "isBalanced" to check whether a given string of parentheses is balanced.
        a. The "isOpenParenthesis" function should call "isBalanced" to check the substring within the parentheses.
        b. The "isBalanced" function should call "isOpenParenthesis" to check if the opening parenthesis has a matching closing parenthesis.
        c. Prompt the user to input a string containing parentheses, and then call the "isBalanced" function to determine and display whether the parentheses are balanced.

- Part 3
    1. **Palindrome Check:** Implement a recursive function to check if a linked list is a palindrome. A linked list is a palindrome if the sequence of its elements is the same when read forward and backward.
    2. **Combination Sum:** Given a set of candidate numbers and a target sum, write a recursive function to find all unique combinations of candidates that sum up to the target. Each number in the candidate set can be used multiple times.
    3. **Nested List Sum:** Write a recursive function to find the sum of all elements in a nested list of integers. The nested list can contain integers or other nested lists.
    4. **Subset Sum:** Write a recursive function to find if there exists a subset of a given set of integers that adds up to a given target sum. The function should return True if such a subset exists; otherwise, return False.
    5. **Counting Paths:** Given a 2D grid of size m x n, write a recursive function to count all possible unique paths from the top-left corner (0, 0) to the bottom-right corner (m-1, n-1). You can only move right or down in the grid.
    6. **Maze Solver:** Given a maze represented as a 2D array where 0 represents an open cell and 1 represents a blocked cell, write a recursive function to find a path from the start point to the end point, if one exists. You can move in any direction (up, down, left, or right) but not diagonally.
    7. **Expression Evaluation:** Write a recursive function to evaluate simple arithmetic expressions represented as strings. The expressions can contain addition, subtraction, multiplication, and division.
    8. **Binary Search:** Implement a recursive function for binary search in a sorted list. Given a sorted list of integers and a target value, write a function to determine if the target value exists in the list and return its index if found, or -1 if not found.
    9. **Binary Tree Traversals:** Implement recursive functions for three types of binary tree traversals: Pre-order, In-order, and Post-order. Traverse a binary tree and print its elements in the specified order.