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.

- Part 1
    1. **Factorial Calculation:** Write a recursive function to calculate the factorial of a given positive integer n. The factorial of n, denoted as n!, is the product of all positive integers less than or equal to n.
    2. **Fibonacci Sequence:** Implement a recursive function to find the n-th number in the Fibonacci sequence. The Fibonacci sequence is a series of numbers where each number is the sum of the two preceding ones. The sequence starts with 0 and 1, i.e., 0, 1, 1, 2, 3, 5, 8, 13, 21, ...
    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.
    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. **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.
    7. **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.
    8. **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.

- 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. **Insertion:** Write a recursive function to insert a new node with a given value at the end of the linked list. The function should traverse the list until it reaches the last node (i.e., a node with a null reference) and then insert the new node.
    4. **Deletion:** Create a recursive function to delete a node with a specific value from the linked list. The function should traverse the list, find the node to delete, and adjust the pointers to skip over the node.
    5. **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.
    6. **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.
    7. **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.
    8. **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.

- Part 3
    1. **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.
    2. **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.
    3. **Sum of Digits:** Write a recursive function to calculate the sum of the digits of a positive integer. For example, the sum of digits for 123 would be 1+2+3 = 6.
    4. **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.
    5. **Expression Evaluation:** Write a recursive function to evaluate simple arithmetic expressions represented as strings. The expressions can contain addition, subtraction, multiplication, and division.
    6. **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.
    7. **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.