LAB 5 : Recursion
[No Submission]

Instructions for students:

- Complete the following methods.
- You may use Java / Python to complete the tasks.
- DO NOT CREATE a separate folder for each task.
- If you are using **JAVA**, then follow the **Java Template**.
- If you are using **PYTHON**, then follow the **Python Template**.

NOTE:

- **YOU CANNOT USE ANY OTHER DATA STRUCTURE OTHER THAN ARRAY AND LINKED LIST.**
- **YOUR CODE SHOULD WORK FOR ANY VALID INPUTS.**

Python List, Negative indexing, slicing and append() is STRICTLY prohibited

TOTAL TASKS: 5

`You can use as many helper methods/functions or add as many parameters as you need. But you can’t use global/instance variables`

Task 1:

A. Write a method/function called **task1A()** that uses a while loop to print from 1 to 10.
B. Convert the previous task into a recursive function called **task1B_recursive()**.
C. Write a method/function called **task1C()** that uses a while loop to print from 1 to N, here N is a user input.
D. Convert the previous task into a recursive function called **task1D_recursive()**.

Task 2: [You can use extra parameters as per your need]

A. Write a method/function called **task2A( )** that **`takes an array in its parameter`** and uses a while loop to print the array elements.

B. Convert the previous task into a recursive function called **task2B_recursive()**.

C. Write a method/function called **task2C( )** that **`takes an array in its parameter`** and **returns the summation of the elements**.

D. Convert the previous task into a recursive function called **task2D_recursive()**.

E. Write a method/function called **task2E( )** that **`takes an array in its parameter.`** The function **returns the subtraction** between the multiplication of odd elements and summation of the even elements.

F. Write a function called **task2F_recursive()** which does the same thing as task2E but utilizes recursion. The **task2F_recursive()** doesn’t have to be recursive if the helper function uses recursion.
`Note:` You may use helper functions for this task. i.e. one helper function may do the summation and another function may do the multiplication then the **task2F_recursive()** would just call them and return the subtraction.

Task 3:

A. Write a method/function called **task3A( )** that **`takes the head of Singly LinkedList in its parameter`** and uses a while loop to print the array elements.

B. Convert the previous task into a recursive function called **task3B_recursive()**.

C. Write a method/function called **task3C( )** that **`takes the head of Singly LinkedList in its parameter`** and **returns the summation of the elements**.

D. Convert the previous task into a recursive function called **task3C_recursive()**.

E. Write a method/function called **task3E( )** that **`takes the head of Singly LinkedList in its parameter .`** The function **returns the subtraction** between the summation of odd elements and multiplication of the even elements.

F. Write a function called **task3F_recursive()** which does the same thing as task3E but utilizes recursion.

`Note:` You may use helper functions for this task

Task 4:

You are **not allowed** to use nodeAt() or create any new linked lists for task4 A or B. The following two tasks are important in order to understand how backtracking works in recursive functions. To understand better try to draw/simulate the recursive calls.

A. Write a method/function called **task4A_recursive( )** that **`takes the head of Singly LinkedList in its parameter`** and prints the elements in the reverse order.

B. Write a method/function called **task4B_recursive( )** that **`takes the head of Singly LinkedList in its parameter`** and reverses the linked list then **returns the new head**.

Task 5:

You are **not allowed** to create any new linked lists for task5 A, B, C.

A. Write a method/function called **findMax_recursive( )** that **`takes the head of a Singly LinkedList in its parameter`** and **returns the maximum number** from the linkedlist. You may use helper functions.

For the two Task5B and Task5C below we'll be simulating nested loops into recursive functions. It's easier to solve these problems if you dedicate one recursive function/method to do the tasks of one loop.

B. Write a method/function called **sortLL_recursive( )** that **`takes the head of Singly LinkedList in its parameter`**. The function would sort the linked list using the ***selection sorting algorithm*** and finally **return the updated head**.

Note: You can just swap the values of the nodes, no need to connect/disconnect nodes. You can also use helper functions and utilize the previous function to solve this.

C. Write a recursive method/function called **findDup( )** that **`takes the head of Singly LinkedList in its parameter`** and then print each value and the indices where the duplicates are found.

Note: You can use helper functions to solve this.

Sample Linked List	Sample Output
10 → 22 → 13 → 20 → 22 → 23 → 10 → 22	10: 6 22: 4, 7 13: No Duplicate 20: No Duplicate 22: 1, 7 23: No Duplicate 10: 0 22: 1, 4

Extra Tasks:

All looping tasks can be converted into recursion. However, not all recursive tasks can be solved using loops. There are certain tasks that can only be solved using a recursive approach. Like the following two tasks.

- Finding whether a singly linked list is a palindrome or not. You're not allowed to use any extra helper functions, no extra data structures or modifying the given data structure.

- Flattening a multi level linked list or nested array. For example,
  Given Array:
  [1, [2, [3, [4], 5], 6], 7, 8, [9, [[10, 11], 12], 13], 14, [15, [16, [17]]]]
  Output Array:
  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]