Lab Exam 3 Preliminary

In this lab, you are going to be familiar with "Struct" and "Linked List" in C. It consists of two parts, which are "Basic Introduction to Struct" and "Linked List Operations".

Note: For convenience, linked lists will be represented as follows in this document:

```
2 -> 3 -> 1 -> 5 -> 1 -> 4
```

Important Specifications

- You can not use any library other than provided libraries as <stdio.h> and <stdlib.h>.
- DO NOT use keywords static.
- You are NOT allowed to use global variables anywhere.
- You are NOT allowed to change the part above the "Do not change the lines above" line.
- Please submit your implementations in the "lab3.c" file.
- You can define your own helper functions.
- Make sure that functions given to you as tasks can be called "as is". DO NOT change their signature.
- Make sure that your solution works with the examples that are given to you in the description first.
- Your codes will be compiled with "-ansi -Wall -g -O0 -pedantic-errors" flags. Please follow ANSI standards otherwise your code will get errors.
- As in the previous labs, a copy of the header file that contains the function prototypes is also provided as "lab3_copy.h":
    - It is an exact copy of lab3.h, which will be used while evaluating your codes.
    - This file is not used in the evaluation, it was shared with you to inform what is in lab3.h file.
    - Any change in this file will have no effect in the run, debug or evaluate commands.
- You can use the "run.c" file for testing your functions with the run and debug commands, modify it however you like. When you click the "run" button, main in the "run.c" will be evaluated. This file will not be used in the evaluation.

1. Basic Introduction to Struct

In this part, you are provided one "Student" struct, you can see the declaration below and in the lab3_copy.h file.

```
typedef struct Student
{
    int studentID;
} Student;
```

Question - 1 (20 Points)

Function:

```
Student** create_student_array(int* student_id_list, int student_count)
```

Explanation:
You are given two parameters, which are ”student_id_list“ and ”student_count“and a ”Student“ struct. You should take studentIDs one by one from the student_id_list array and create a student for each studentID in that list. Then, you should create and return an array whose elements are student pointers (Student*).

Important Note:
The order of the studentIDs in student_id_list and your returned array must be the same. Example: If the student_id_list is [1, 3, 2], then your returned array must be [pointer_to_Student_whose_id_1, pointer_to_Student_whose_id_3, pointer_to_Student_whose_id_2].

Example:

```
    student_id_list[5] = {1, 2, 3, 4, 5}
    student_count = 5
    returned_array = create_student_array(student_id_list, student_count);
    returned_array: [pointer_to_Student_whose_id_1, pointer_to_Student_whose_id_2,
                     pointer_to_Student_whose_id_3, pointer_to_Student_whose_id_4,
                     pointer_to_Student_whose_id_5]
```

2. Linked List Operations

In this part, you are going to implement several functions related to linked lists. Linked list is a data structure in which objects refer to the same kind of object and are linked in a linear sequence.
Each node in the linked list is composed of a value whose type is int and a pointer to the next node. The next pointer of the last node in the linked list should be NULL.
Linked list ”Node“ is defined in lab3.h file, you can see the declaration below and in the lab3_copy.h file.

```
struct Node
{
    int data;
    struct Node *next;
```

```
};
typedef struct Node Node;
```

Question - 2 (10 Points)

Function:

```
Node* insert_head(Node* head, int new_node_data)
```

Explanation:
With the given new_node_data, you should create a new node and add it to the beginning of the given linked list.
You have to return the head of the changed list. You should not create a new list but update the existing list. If the head of the list does not change, after the operation, you can just return the same head value as the given input.

Example 1:

```
list (before): 1 -> 5 -> 3 -> 4
new_node_data = 2
list = insert_head(list, new_node_data);
list (after): 2 -> 1 -> 5 -> 3 -> 4
```

Example 2:

```
list (before): NULL
new_node_data = 3
list = insert_head(list, new_node_data);
list (after): 3
```

Question - 3 (15 Points)

Function:

```
Node* remove_kth_node(Node* head, int index)
```

Explanation:
You should remove the node at the given index and connect the previous and next nodes of the removed node. Range of "index" variable is [0, length_of_list-1].
You have to return the head of the changed list. You should not create a new list but update the existing list. If the head of the list does not change, after the operation, you can just return the same head value as the given input.

Note:
If there is only one node in the given list, as you can understand, the new list must be empty.

Example 1:

```
list (before): 1 -> 5 -> 3 -> 4
index = 1
list = remove_kth_node(list, index);
list (after): 1 -> 3 -> 4
```

Example 2:

```
list (before): 1 -> 5 -> 3 -> 4
index = 0
list = remove_kth_node(list, index);
list (after): 5 -> 3 -> 4
```

Example 3:

```
list (before): 1
index = 0
list = remove_kth_node(list, index);
list (after): NULL
```

Question - 4 (15 Points)

Function:

```
int find_occurrences(Node* head, int data)
```

Explanation:

You should search the entire list and return how many nodes are there whose data equals to given data.

Example 1:

```
list: 1 -> 5 -> 3 -> 5 -> 4
data = 5
occurrences = find_occurrences(list, data);
occurrences: 2
```

Example 2:

```
list: 1 -> 5 -> 3 -> 5 -> 4
data = 2
occurrences = find_occurrences(list, data);
occurrences: 0
```

Example 3:

```
list: NULL
data = 2
occurrences = find_occurrences(list, data);
occurrences: 0
```

Question - 5 (20 Points)

Function:

```
Node* copy_to_head(Node* head, int index)
```
Explanation:
After the node at the given index, you should copy the rest of the linked list (including the node at the given index) and paste it to the head of the list. In other words, you should copy the sub linked list between the node at the given index and the last node of the list (including them), then paste it to the head of the list. Range of "index" variable is [0, length_of_list-1].
You have to return the head of the changed list. You should not create a new list but update the existing list. If the head of the list does not change, after the operation, you can just return the same head value as the given input.

Example 1:

```
list (before): 1 -> 5 -> 3 -> 4
index = 2
list = copy_to_head(list, index);
list (after): 3 -> 4 -> 1 -> 5 -> 3 -> 4
```
Example 2:

```
list (before): 1 -> 5 -> 3 -> 4
index = 0
list = copy_to_head(list, index);
list (after): 1 -> 5 -> 3 -> 4 -> 1 -> 5 -> 3 -> 4
```
Example 3:

```
list (before): 1
index = 0
list = copy_to_head(list, index);
list (after): 1 -> 1
```

Question - 6 (20 Points)

Function:

```
Node* sum_consecutive_pairs(Node* head)
```
Explanation:
You should create and return a new list by summing the data values of the consecutive node pairs of the given list. You can modify the given list whatever you like.
Example: If the given list is 1 -> 2 -> 3, the node count of the new list must be 2. Data of the first node in the new list must be 1 + 2 = 3 and data of the second node in the new list is 2 + 3 = 5. Then, the new list must be 3 -> 5.

Important Note:
The node count of the given list will always be greater than or equal to 2.

Example 1:

```
list: 1 -> 5 -> 3 -> 4
new_list = sum_consecutive_pairs(list);
new_list: 6 -> 8 -> 7
```

Example 2:

```
list: 1 -> 4 -> 1
new_list = sum_consecutive_pairs(list);
new_list: 5 -> 5
```

Example 3:

```
list: 2 -> 5
new_list = sum_consecutive_pairs(list);
new_list: 7
```