BBM-201 Data Structures Exercise: Lists (3)

Instructor: Prof. Dr. Emirhan Yalçın

Topics: Linked List, Array List

Name:				Id:				
Surname:				Section:				
Total 100 Points								
Question	1 (12pt)	2 (12pt)	3 (16pt)	4 (20pt)	5 (14pt)	6 (12pt)	7 (14pt)	Total: (100pt)
Deserved:								

- This exam consist of 7 list questions. You can find more complex questions on this papers compare to first two exersizes, that were mainly focusing on basic list operations.
- Some questions are taken from **Leetcode**. You can find the question numbers and links in the question text.
- Since linked list is an important data structure, being base for almost all the other structures, you are adviced to practice this data structure well.
- In order to provide active learning, you are adviced to solve this questions with a time limit and solve as you are in a real exam, with detailed explanations.
- Good luck.

1. *(12 points)* **Merge Lists:** Implement the **merge** method for list data structure that merges to sorted lists.

   (a) Implement the merge method for two sorted Linked Lists, heads of the lists are given to you. Merge the lists and return the head of new list.

   (Leetcode Problem 21 Link)

```
ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
```

(b) Implement merge method for merging two dynamically allocated and sorted arrays. Return pointer of merged array.

```
int* mergeTwoLists(int* arr1, int size1, int* arr2, int size2) {
```

2. *(12 Points)* **Reverse Linked List:** Given the head of the singly linked list, implement a method that reverses the list and returns the new head.

   (Leetcode Problem 206)

```
ListNode* reverseList(ListNode* head) {
```

3. *(16 Points)* **Stack Implementation Using List ADT:**

What is a stack?

Stack is a speacial type of data structure which is often used in computer science and algorithms. It has some common features with lists. What is special for a stack is that, insert and delete operations are always made from end of the stack. It follows LIFO (Last in First out) principle. There are 4 fundamental methods of a stack:

- push(**int** x): Pushes new element x to end of the stack.
- pop( ) ⇒ **int**: Removes the top (last inserted element) of the stack and returns it.
- top( ) ⇒ **int** : Returns top (last inserted element) of the stack, without removing it.
- isEmpty( ) ⇒ **boolean** : Return **true** if stack is empty.

Stack makes all this operations in **O(1)** time complexity.

Your question is implementing the stack data structure using the list ADT attributes and methods. Implement Stack class and 4 methods above using C++, write your solution to the text page.

You have seen different types of list implementations such as Singly and Doubly Linked List, ArrayList, Circular lists etc. Which type of list will you use for implementing stack? Analize your implementation for its memory and time complexities, show with reasons that your implementation does all the operation in O(1) time complexity.

```
class Stack {

}
```

4. *(20 Points)* **Queue Implementation Using List ADT:**

What is a queue?

Queue is a speacial type of data structure which is often used in algorithms. What is special about the queue is insertions are made in end of the queue, while deletions made from the head. Basically queue has FIFO (First in First out) principle. Queue has 5 fundamental methods:

- enqueue(**int** x): Pushes element to the end of the queue.
- dequeue( ) ⇒ **int**: Removes the most-front element of the queue and returns it.
- head( ) ⇒ **int**: Returns the head element of the queue without removing it.
- tail( ) ⇒ **int**: Returns the very last element of the queue without removing it.
- isEmpty( ) ⇒ **boolean**: Returns **true** if queue is empty.

Queue makes all this operations in **O(1)** time complexity.

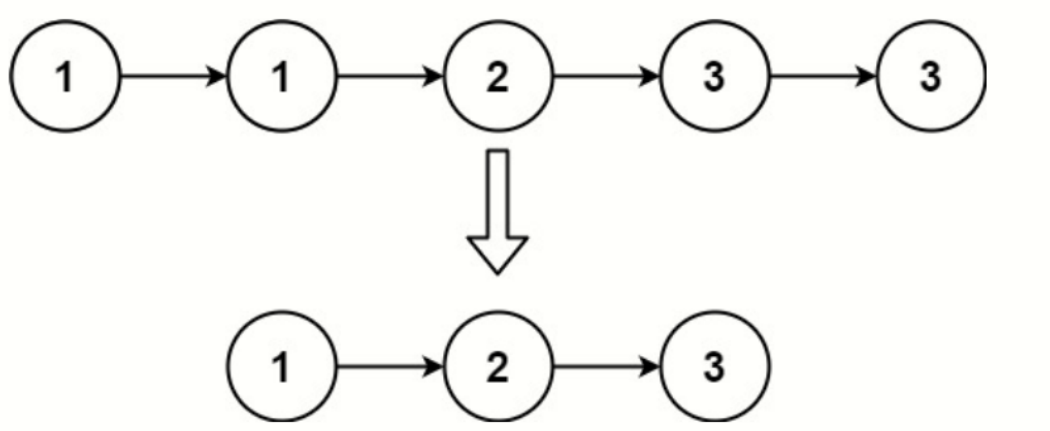
Your question is designing a queue using list ADT. Create a Queue class implement those 5 methods using C++. Write your solution to the next page. Which type of list you will use? Analize your implementation for its memory usage and time complexities, show and proof with explanations that all the methods that you implemented runs in O(1) time complexities.

```
class Queue {

}
```

5. *(14 Points)* **Remove Duplicates from Linked List:** You are given head of a sorted singly linked list. Write a method that removes the duplicates from the linked list and returns the new head of the list. Note that a duplicate can occur more than two times.

   (Leetcode Problem 83 Link)



   (a) Implement the method using iteration.

```
Node* removeDuplicates(ListNode* head) {
```

(b) Implement the method using recursion.

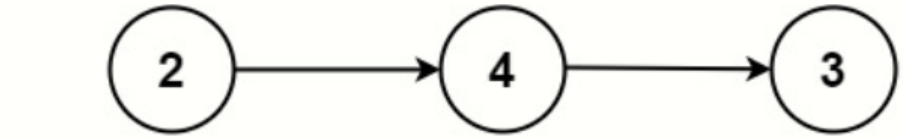
```
Node* removeDuplicates(ListNode* head) {
```

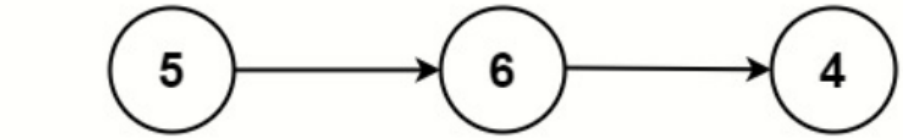
6. *(12 Points)* **Add Two Numbers Represented with Linked List:** You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list.

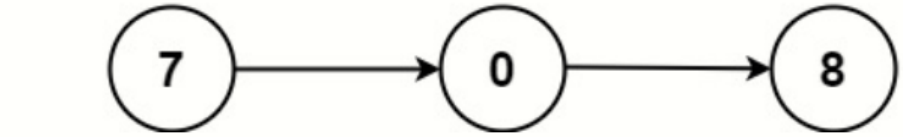
   You may assume the two numbers do not contain any leading zero, except the number 0 itself. But note that two numbers does not have to have same number of digits.

   (Leetcode Problem 2 Link)

Example 1:








```
Input: l1 = [2,4,3], l2 = [5,6,4]
Output: [7,0,8]
Explanation: 342 + 465 = 807.
```

Constraints:

- The number of nodes in each linked list is in the range `[1, 100]`.
- `0 <= Node.val <= 9`
- It is guaranteed that the list represents a number that does not have leading zeros.

```
ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
```

7. *(14 Points)* **Delete a Linked List Using Recursion:** Given the head of a singly linked list, implement a recursive method that deletes all the nodes. Allocate memory by deleting all the nodes including the head.

```
void deleteLinkedList(Node* head) {
```