

BBM-201 Data Structures Exercise: Lists (1)

Instructor: Prof. Dr. Emirhan Yalçın

Topics: Linked List, Array List

November 1, 2023

Name:				Id:			
Surname:				Section:			
Total 100 Points							
Question	1 (15pt)	2 (16pt)	3 (24pt)	4 (12pt)	5 (25pt)	6 (18pt)	Total: (100pt)
Deserved:							

- This document is not a graded examination, it is just an exercise for you to practice and understand basics of list data structure better with solving some simple algorithmic questions. But this paper is designed like a real exam paper and you are encouraged to solve this questions with a time limit as you are in an exam in order to provide active learning.
- This exercise mainly focuses on basic list operation and comparison of *Linked List* and *Array List*. Questions are mostly about fundamental operations of lists and you should learn them very well in order to solve more complicated problems in future.
- Another exercises with relatively more complicated questions will be published with you later.
- You can solve question using **Java** or **C++** programming language. Solving questions first with Java and after that converting the solution to C++ is adviced for both reviewing your solution for better understanding and since debugging with C++ may be relatively harder especially if you are a beginner.
- You are adviced to solve this questions again and again until they become really easy for to implement solutions, since List is a important data structure and actually a base for others. You should be able to implement solutions almost automatically.

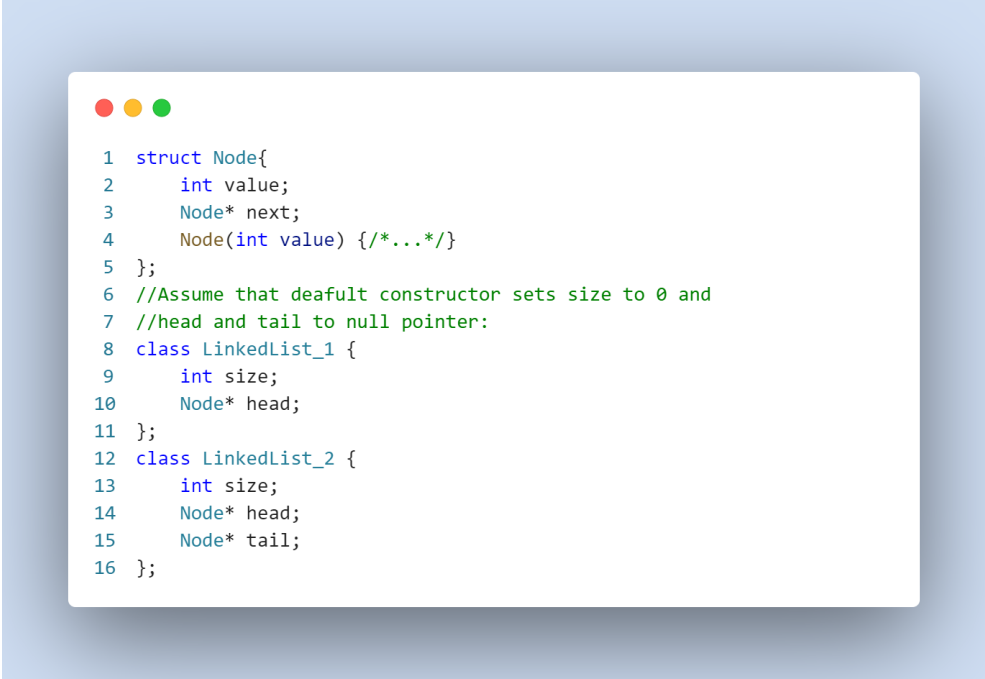
1. (15 points) **List Types Comparison:** List data structure can be implemented both array based and node based. Java programming language has both types of list classes built-in. Java programming language, there is two type of List classes: ArrayList and LinkedList.

(a) What is the main difference between a ArrayList and a LinkedList? Basically explain the main structure of both data structures.

(b) When we compare both type of lists, when it is usefull to choose ArrayList over LinkedList? What is the advantage of using it?

(c) For which types of operation(s) it is better to use LinkedList instead of ArrayList? Explain with reasons.

2. (16 points) **Insert Method Implementations:** Below you can see two types of implementations of a Singly Linked List. While first implementation keeps pointer of only head node, second implementation keeps pointer of both head and tail node of the list.



```
1 struct Node{
2     int value;
3     Node* next;
4     Node(int value) {/*....*/}
5 };
6 //Assume that default constructor sets size to 0 and
7 //head and tail to null pointer:
8 class LinkedList_1 {
9     int size;
10    Node* head;
11 };
12 class LinkedList_2 {
13     int size;
14     Node* head;
15     Node* tail;
16 };
```

- (a) Implement **insertAtHead(int value)** and **insertAtTail(int value)** methods for the first implementation, which only keeps pointer to the head of the list. What is the time complexity for your methods?

- (b) Implement same methods for the second implementation, considering both head and tail pointers. What is the time complexity of the methods for this implementation?

3. (24 points) **Get Methods for Lists:** Below you can see basic structure of a LinkedList and ArrayList. If index is invalid, return -1. ($0 \leq \text{index} < \text{size}$)

```
1 class LinkedList {
2     int size;
3     Node* head;
4     LinkedList() :
5         size(0), head(nullptr) {}
6 };
7 class ArrayList {
8     int size;
9     int* main_array;
10    ArrayList(int maxSize){
11        this->main_array = new int[maxSize];
12        size = 0;
13    }
14 };
```

- (a) Implement following methods for LinkedList. Write the time complexities of your methods and explain why.

getHeadNode() \Rightarrow *Node*

getTailNode() \Rightarrow *Node*

getAtIndex(*int* index) \Rightarrow *int*

- (b) Implement following methods for ArrayList implementation. Again, mention about time complexities of your methods.

getFirst() $\Rightarrow int$

getLast() $\Rightarrow int$

getAtIndex(*int* index) $\Rightarrow int$

4. (12 points) **Reverse Array List:** Considering the same implementation in Question 3, write two methods that reverses an array list.

(a) Implement the reverse method for Array List using iteration.

(b) Implement the same method using recursion.

5. (25 points) **Insert At Index** Considering the same list implementations in question 3, implement **insertAtIndex**(*int index*) method for linked list and array list respectively, which can be used for inserting elements to the middle of the list, to a specific position. Your method should care about invalid indexes. ($0 \leq \mathbf{index} \leq \mathit{size}$) (Note: if index equals size of the list, new item will be inserted at the end of the list. You are allowed to use pre-defined methods in former questions.)

(a) (*10 points*) Implement for Linked List:

(b) (*10 points*) Implement for Array List:

(c) (*5 points*) Analyse and compare two methods considering their time complexity and number of operations. Which method is more efficient? Use Big-oh and Tilda notation while mentioning about number of operations.

6. (18 points) **Delete Head and Tail of a Linked List:**

- (a) Implement a method names **popHead()** \Rightarrow **int** that deletes the head node of the Linked List. Consider the implementation in Question 3, in which you have the pointer of only the head node. Your method should return the deleted value. If the list is empty, return -1.
- (b) Implement **popTail()** \Rightarrow **int** method, which removes the last element of the list and returns its value. If the list is empty, return -1.