

# *Data Structures*

## SLL Homework 1

**Mostafa S. Ibrahim**

*Teaching, Training and Coaching for more than a decade!*

*Artificial Intelligence & Computer Vision Researcher*

*PhD from Simon Fraser University - Canada*

*Bachelor / Msc from Cairo University - Egypt*

*Ex-(Software Engineer / ICPC World Finalist)*



# Tip

- For ALL homework exercises in this course, you must:
  - Compute time order
  - Compute memory order
    - Skip the effect of the debug\_data items. They are for educational purposes

# Problem #1: Insert front



- We implemented insert\_end in the lecture
- We want to be able to insert front as shown:
- Write down your own test cases
  - Think in special cases
  - Empty list
  - Single item list
  - 2 items list
  - N items list

```
lst = LinkedList([4])
lst.insert_front(3)
lst.insert_front(2)
lst.insert_front(1)

lst.debug_print_existing_nodes()
result = str(lst)
expected = '1, 2, 3, 4'
```

## Problem #2: Delete front



- The opposite of insert front
  - Below, node 6 will be deleted

```
lst = LinkedList([6, 10, 8, 15])  
lst.delete_front()  
  
lst.debug_print_existing_nodes()  
result = str(lst)  
expected = '10, 8, 15'
```

## Problem #3: Get nth from the back

- We already implemented `get_nth(n)`
- Now implement: `get_nth_back(int n)`
  - Given a 1-based position, find the node from the back and return it
  - Return None if such position doesn't exist

```
lst = LinkedList([6, 10, 8, 15])  
result = str(lst.get_nth_back(3))  
  
lst.debug_print_existing_nodes()  
expected = '10'
```

## Problem #4: Is Identical?

- Develop a function that checks if 2 lists have **identical data**:
  - Each list must be the **same length**
  - The **value** of a node in one list must match the value of its **corresponding** node in the other list
- In this coding, assume *you can't use length variable or compute the length of linked list explicitly*

```
lst1 = LinkedList([1, 2])  
lst2 = LinkedList([1, 2, 3])  
result = str(lst1.is_identical_data(lst2))  
  
expected = 'False'
```

# Problem #5: Linked List without tail/length!

- Assume we'll implement our linked list to have only a head pointer - and no tail
- Implement and test these 2 methods
  - **add\_element**: this simply adds a new element to our current collection of numbers:  $O(1)$ 
    - Tip: data will be **reversed**
  - **get\_tail** will retrieve the last node in our list

```
class LinkedList:
    def __init__(self, initial_values=None):
        self.head = None

        if initial_values:
            for value in initial_values:
                self.add_element(value)

    def add_element(self, value):
        pass

    def get_tail(self):
        pass
```

*“Acquire knowledge and impart it to the people.”*

*“Seek knowledge from the Cradle to the Grave.”*