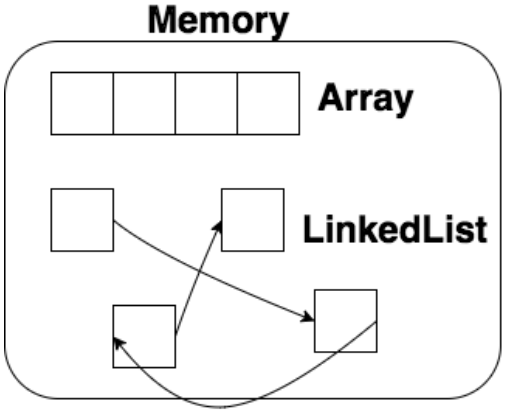
CSCI-SHU 210 Data Structures

Recitation6 Worksheet Linked Lists

You have a series of tasks in front of you. Complete them! Everyone should code on their own computer, but you are encouraged to talk to others, and seek help from each other and from the TA/LAs.

Important for this week:

- **Understand "reference based" data structure.**
  - **And "reference based" data structure's coding style!**



- **Understand the many-faced Linked List data structure. Difference between**
  - **Single ended vs double ended linked lists**

```
def __init__(self):
    self.head = None
    self.size = 0
```

```
def __init__(self):
    self.head = None
    self.tail = None
    self.size = 0
```

  - **Singly linked vs doubly linked lists**

```
def __init__(self, element, next):
    self._element = element
    self._next = next
```

```
def __init__(self, element, prev, next):
    self._element = element
    self._prev = prev
    self._next = next
```

  - **Linked Lists with empty sentinel vs Linked Lists without empty sentinel**

```
def __init__(self):
    self.header = self._Node(None, None, None)
    self.trailer = self._Node(None, None, None)
    self.header._next = self.trailer
    self.trailer._prev = self.header
    self.size = 0
```

```
def __init__(self):
    self.head = None
    self.tail = None
    self.size = 0
```

- **Know the advantages of Linked List data structure.**
  - **Fast (O(1)) insertion/deletion, if we have the reference.**

- **Can optimize solutions to problems using Linked List data structure.**

Part 1: Implement Deque using `Double ended doubly linked list`.

We've already implemented stack using `Single ended singly linked list`; Why?

We've also implemented queue using `Double ended singly linked list`. Why?

Let's now implement **Deque** using `Double ended doubly linked list`. "Double ended" means the linked list has a reference to the head node, and a reference to the tail node.

Doubly linked means each node has two linkages. "_next" and "_prev".

Your task 1: Implement class `LinkedDeque`. You should complete those:

`first(self)`	`last(self)`
`delete_first(self)`	`delete_last(self)`
`add_first(self, e)`	`add_last(self, e)`

Part 2: Single Linked List Exercises.

Your task 2: Implement function `return_max(self)` in `SingleLL.py`

Traverse the single linked list and return the maximum element stored with in the linkedlist.

Your task 3: Implement function `__iter__(self)` in `SingleLL.py`

Generate a forward iteration of the elements from self linkedlist.
Remember to use keyword "yield"!

Your task 4: Implement function `insert_after_kth_index(self, k, e)` in `SingleLL.py`

Insert element e (as a new node) after kth indexed node in self linkedlist.

For example,

L1: 11-->22-->33-->44-->None

L1.insert_after_kth_position(2, “Hi”) # 33 is the index 2.

L1: 11-->22-->33-->”Hi”-->44-->None

Part 3: Double Linked List Exercises.

Your task 5: Implement function `split_after(self, index)` in `DoubleLL.py`

After called, split self DoubleLinkedList into two separate lists.

Self list contains first section, return a new list that contains the second section.

For example,

L1: head<-->1<-->2<-->3<-->4-->tail

L2 = L1.split_after(2)

L1: head<-->1<-->2<-->3-->tail

L2: head<-->4-->tail

Your task 6: Implement function `merge(self, other)` in `DoubleLL.py`

This function adds other DoubleLinkedList to the end of self DoubleLinkedList. After merging, other list becomes empty.

For example,

L1: head<-->1<-->2<-->3-->tail

L2: head<-->4-->tail

L1.merge(L2)

L1: head<-->1-->2-->3<-->4-->tail

L2: head<-->tail