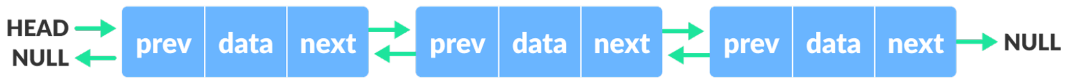
DoublyLL

February 25, 2021

0.1 Node class

[2]:
```
class Node:
    def __init__(self, data):
        self.item = data
        self.nref = None
        self.pref = None
```



0.2 Doubly LL Class

[3]:
```
class DoublyLinkedList:
    def __init__(self):
        self.start_node = None


    def insert_in_emptylist(self, data):
        if self.start_node is None:
            new_node = Node(data)
            self.start_node = new_node
        else:
            print("list is not empty")


    def insert_at_start(self, data):
        if self.start_node is None:
            new_node = Node(data)
            self.start_node = new_node
            print("node inserted")
            return
        new_node = Node(data)
        new_node.nref = self.start_node
        self.start_node.pref = new_node
        self.start_node = new_node
```

```python
def insert_at_end(self, data):
    if self.start_node is None:
        new_node = Node(data)
        self.start_node = new_node
        return
    n = self.start_node
    while n.nref is not None:
        n = n.nref
    new_node = Node(data)
    n.nref = new_node
    new_node.pref = n


def insert_after_item(self, x, data):
    if self.start_node is None:
        print("List is empty")
        return
    else:
        n = self.start_node
        while n is not None:
            if n.item == x:
                break
            n = n.nref
        if n is None:
            print("item not in the list")
        else:
            new_node = Node(data)
            new_node.pref = n
            new_node.nref = n.nref
            if n.nref is not None:
                n.nref.prev = new_node
            n.nref = new_node
def insert_before_item(self, x, data):
    if self.start_node is None:
        print("List is empty")
        return
    else:
        n = self.start_node
        while n is not None:
            if n.item == x:
                break
            n = n.nref
        if n is None:
            print("item not in the list")
        else:
            new_node = Node(data)
```

```
                new_node.nref = n
                new_node.pref = n.pref
                if n.pref is not None:
                    n.pref.nref = new_node
                n.pref = new_node

    def delete_at_start(self):
        if self.start_node is None:
            print("The list has no element to delete")
            return
        if self.start_node.nref is None:
            self.start_node = None
            return
        self.start_node = self.start_node.nref
        self.start_prev = None;

    def delete_at_end(self):
        if self.start_node is None:
            print("The list has no element to delete")
            return
        if self.start_node.nref is None:
            self.start_node = None
            return
        n = self.start_node
        while n.nref is not None:
            n = n.nref
        n.pref.nref = None


    def traverse_list(self):
        if self.start_node is None:
            print("List has no element")
            return
        else:
            n = self.start_node
            while n is not None:
                print(n.item , " ")
                n = n.nref

    def reverse_linked_list(self):
        pass
```

[4]:
```
new_linked_list = DoublyLinkedList()
```

[7]:
```
new_linked_list.insert_in_emptylist(228)
```

```
list is not empty
```

[8]:
```
new_linked_list.traverse_list()
```

```
228
```

[9]:
```
new_linked_list.insert_at_start(10)
new_linked_list.insert_at_start(5)
new_linked_list.insert_at_start(18)
```

[10]:
```
new_linked_list.traverse_list()
```

```
18
5
10
228
```

[11]:
```
new_linked_list.insert_at_end(29)
new_linked_list.insert_at_end(39)
new_linked_list.insert_at_end(49)
```

[12]:
```
new_linked_list.traverse_list()
```

```
18
5
10
228
29
39
49
```

[13]:
```
new_linked_list.delete_at_start()
```

[14]:
```
new_linked_list.traverse_list()
```

```
5
10
228
29
39
49
```

Reversing a Doubly Linked List- Hints To reverse a doubly linked list, you basically have to perform the following operations:

```
The next reference of the start node should be set none because the first node will become the
The previous reference of the last node should be set to None since the last node will become t
The next references of the nodes (except the first and last node) in the original list should b
```

[ ]:
```

```