Experiment No. 6
Program for data structure using built in function for link list, stack and queues
Date of Performance:
Date of Submission:

Experiment No. 6

Title: Program for data structure using built in function for link list, stack and queues

Aim: To study and implement data structure using built in function for link list, stack and queues

Objective: To introduce data structures in python

Theory:

Stacks -the simplest of all data structures, but also the most important. A stack is a collection of objects that are inserted and removed using the LIFO principle. LIFO stands for "Last In First Out". Because of the way stacks are structured, the last item added is the first to be removed, and vice-versa: the first item added is the last to be removed.

Queues – essentially a modified stack. It is a collection of objects that are inserted and removed according to the FIFO (First In First Out) principle. Queues are analogous to a line at the grocery store: people are added to the line from the back, and the first in line is the first that gets checked out – BOOM, FIFO!

Linked Lists

The Stack and Queue representations I just shared with you employ the python-based list to store their elements. A python list is nothing more than a dynamic array, which has some disadvantages.

The length of the dynamic array may be longer than the number of elements it stores, taking up precious free space.

Insertion and deletion from arrays are expensive since you must move the items next to them over

Using Linked Lists to implement a stack and a queue (instead of a dynamic array) solve both of these issues; addition and removal from both of these data structures (when implemented with a linked list) can be accomplished in constant O(1) time. This is a HUGE advantage when dealing with lists of millions of items.

Linked Lists – comprised of 'Nodes'. Each node stores a piece of data and a reference to its next and/or previous node. This builds a linear sequence of nodes. All Linked Lists store a head, which is a reference to the first node. Some Linked Lists also store a tail, a reference to the last node in the list.

CODE:

```
class Node:
    def __init__(self, data=None):
        self.data = data
        self.next = None


class Stack:
    def __init__(self):
        self.top = None

    def push(self, data):
        new_node = Node(data)
        new_node.next = self.top
```

```
        self.top = new_node

    def pop(self):
        if self.top is None:
            print("Stack is empty")
            return None
        data = self.top.data
        self.top = self.top.next
        return data

    def peek(self):
        if self.top is None:
            print("Stack is empty")
            return None
        return self.top.data

    def is_empty(self):
        return self.top is None


class Queue:
    def __init__(self):
        self.front = None
```

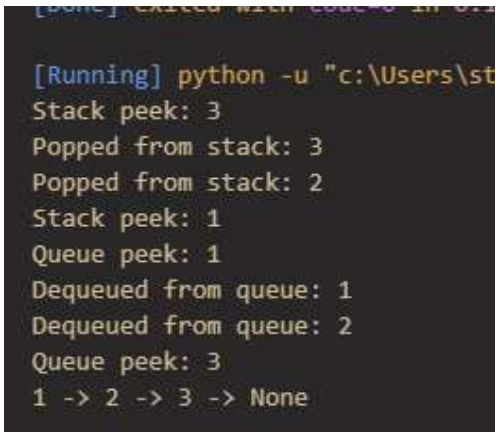
```
    self.rear = None


def enqueue(self, data):
    new_node = Node(data)
    if self.rear is None:
        self.front = new_node
        self.rear = new_node
    else:
        self.rear.next = new_node
        self.rear = new_node


def dequeue(self):
    if self.front is None:
        print("Queue is empty")
        return None
    data = self.front.data
    self.front = self.front.next
    if self.front is None:
        self.rear = None
    return data


def peek(self):
    if self.front is None:
```

```
            print("Queue is empty")
            return None
        return self.front.data

    def is_empty(self):
        return self.front is None


# Testing stack
stack = Stack()
stack.push(1)
stack.push(2)
stack.push(3)
print("Stack peek:", stack.peek())
print("Popped from stack:", stack.pop())
print("Popped from stack:", stack.pop())
print("Stack peek:", stack.peek())

# Testing queue
queue = Queue()
queue.enqueue(1)
queue.enqueue(2)
queue.enqueue(3)
```

```
print("Queue peek:", queue.peek())
print("Dequeued from queue:", queue.dequeue())
print("Dequeued from queue:", queue.dequeue())
print("Queue peek:", queue.peek())


class Node:
    def __init__(self, data):
        self.data = data
        self.next = None



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

    def append(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
            return
        last_node = self.head
        while last_node.next:
            last_node = last_node.next
```

```
        last_node.next = new_node


    def display(self):
        current = self.head
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")




if __name__ == "__main__":

    linked_list = LinkedList()



    linked_list.append(1)
    linked_list.append(2)
    linked_list.append(3)



    linked_list.display()
```

OUTPUT:



Conclusion: Data structures python has been studied and implemented.