Name- Harshit Kumar
Subject- Data Structure Using python
Reg.no- 21ETMC412011
Section- Mathematics and Computing (M&C)
Department- Cse

Total Marks: 12 **Submission Date: 31/03/2023**

Assignment Problems(2 marks each)

1. Implement the stack Data Structure using a fixed capacity array (underlying storage).
2. Implement the singly linked list (instance of Singly list keep reference to head, tail and size).
3. Implement Deque data structure(time complexity of each method must be O(1)) using array.
4. Implement Deque data structure(time complexity of each method must be O(1)) using doubly linked list.
5. Implement stack data structure using two queues.
6. Implement queue data structure using two stacks.

Note: Paste the screen-shots of code and output of each problem

P.T.O

[Question-1]

main.py > ...

```python
class Stack:
    def __init__(self, size=10):
        self._stack = []
        self._size = size

    def is_empty(self):
        return len(self._stack) <= 0

    def push(self, data):
        if len(self._stack) >= self._size:
            raise Exception('Stack overflow')
        else:
            self._stack.append(data)

    def pop(self):
        if len(self._stack) <= 0:
            raise Exception('Stack underflow')
        else:
            return self._stack.pop()

    def peek(self):
        if len(self._stack) <= 0:
            raise Exception('Stack underflow')
        else:
            return self._stack[-1]

    def length(self):
        return len(self._stack)

stack = Stack(5)
stack.push(5)
stack.push(2)
stack.push(3)
stack.push(9)
stack.push(6)
print(stack.peek())
print(stack.pop())
print(stack.length())
print(stack.peek())
```

Output-

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
6
6
4
9
PS C:\Users\suruc\PycharmProjects\firstfrog>
```

P.T.O

[Question-2]

main.py > SinglyLinkedList

```python
class ListNode:
    def __init__(self, val=None):
        self.val = val
        self.next = None

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

    def is_empty(self):
        return self.size == 0

    def append(self, val):
        node = ListNode(val)
        if self.is_empty():
            self.head = node
        else:
            self.tail.next = node
        self.tail = node
        self.size += 1

    def prepend(self, val):
        node = ListNode(val)
        if self.is_empty():
            self.tail = node
        else:
            node.next = self.head
        self.head = node
        self.size += 1

    def insert_after(self, prev_node, val):
        node = ListNode(val)
        node.next = prev_node.next
        prev_node.next = node
        if prev_node == self.tail:
            self.tail = node
        self.size += 1
```

```python

    def delete(self, val):
        if self.is_empty():
            raise Exception('List is empty')
        if self.head.val == val:
            self.head = self.head.next
            if self.head is None:
                self.tail = None
            self.size -= 1
            return
        node = self.head
        while node.next is not None:
            if node.next.val == val:
                if node.next == self.tail:
                    self.tail = node
                node.next = node.next.next
                self.size -= 1
                return
            node = node.next
        raise Exception('Value not found in list')
linked_list = SinglyLinkedList()

linked_list.append(1)
linked_list.append(2)
linked_list.prepend(0)
linked_list.insert_after(linked_list.head.next, 1.5)
linked_list.delete(0)
linked_list.delete(2)

print('List size:', linked_list.size)
node = linked_list.head
while node is not None:
    print(node.val)
    node = node.next
```

Output-

```
List size: 2
1
1.5
PS C:\Users\suruc\PycharmProjects\firstfrog>
```

[Question-3]

main.py

```python
class Deque:
    def __init__(self, capacity):
        self.capacity = capacity
        self.items = [None] * capacity
        self.front = 0
        self.rear = 0
        self.size = 0

    def is_empty(self):
        return self.size == 0

    def is_full(self):
        return self.size == self.capacity

    def push_front(self, item):
        if self.is_full():
            raise Exception('Deque is full')
        self.front = (self.front - 1 + self.capacity) % self.capacity
        self.items[self.front] = item
        self.size += 1

    def push_rear(self, item):
        if self.is_full():
            raise Exception('Deque is full')
        self.items[self.rear] = item
        self.rear = (self.rear + 1) % self.capacity
        self.size += 1

    def pop_front(self):
        if self.is_empty():
            raise Exception('Deque is empty')
        item = self.items[self.front]
        self.front = (self.front + 1) % self.capacity
        self.size -= 1
        return item

    def pop_rear(self):
        if self.is_empty():
            raise Exception('Deque is empty')
        self.rear = (self.rear - 1 + self.capacity) % self.capacity
        item = self.items[self.rear]
        self.size -= 1
        return item
```

```python
deque = Deque(5)

deque.push_front(1)
deque.push_front(0)
deque.push_rear(2)
deque.push_rear(3)
deque.pop_front()
deque.pop_rear()

print('Deque size:', deque.size)
while not deque.is_empty():
    print(deque.pop_front())
```

Output-

```
Deque size: 2
1
2
PS C:\Users\suruc\PycharmProjects\firstfrog>
```

[Question-4]

main.py > ...

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

class Deque:
    def __init__(self):
        self.front = None
        self.rear = None

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

    def add_front(self, data):
        new_node = Node(data)
        if self.is_empty():
            self.front = self.rear = new_node
        else:
            new_node.next = self.front
            self.front.prev = new_node
            self.front = new_node

    def add_rear(self, data):
        new_node = Node(data)
        if self.is_empty():
            self.front = self.rear = new_node
        else:
            new_node.prev = self.rear
            self.rear.next = new_node
            self.rear = new_node

    def remove_front(self):
        if self.is_empty():
            return None
        front_data = self.front.data
        if self.front == self.rear:
            self.front = self.rear = None
        else:
            self.front = self.front.next
            self.front.prev = None
        return front_data

    def remove_rear(self):
        if self.is_empty():
            return None
        rear_data = self.rear.data
        if self.front == self.rear:
            self.front = self.rear = None
        else:
            self.rear = self.rear.prev
            self.rear.next = None
        return rear_data

    def peek_front(self):
        return self.front.data if self.front else None

    def peek_rear(self):
        return self.rear.data if self.rear else None

dq = Deque()
dq.add_front(1)
dq.add_front(2)
dq.add_rear(3)
dq.add_rear(4)
print(dq.peek_front())
print(dq.peek_rear())
print(dq.remove_front())
print(dq.remove_rear())
print(dq.peek_front())
print(dq.peek_rear())
```

Output-

```
2
4
1
3
PS C:\Users\suruc\PycharmProjects\firstfrog>
```

[Question-5]

main.py > ...

```python
class Stack:
    def __init__(self):
        self.queue1 = []
        self.queue2 = []

    def push(self, item):
        self.queue1.append(item)

    def pop(self):
        if not self.queue1:
            return None

        while len(self.queue1) > 1:
            self.queue2.append(self.queue1.pop(0))

        popped_item = self.queue1.pop(0)

        self.queue1, self.queue2 = self.queue2, self.queue1

        return popped_item

    def peek(self):
        if not self.queue1:
            return None

        while len(self.queue1) > 1:
            self.queue2.append(self.queue1.pop(0))

        peeked_item = self.queue1[0]

        self.queue2.append(self.queue1.pop(0))

        self.queue1, self.queue2 = self.queue2, self.queue1

        return peeked_item

    def is_empty(self):
        return not bool(self.queue1)

    def size(self):
        return len(self.queue1)


s = Stack()
s.push(1)
s.push(2)
s.push(3)
print(s)
print(s.pop())
print(s.peek())
print(s.pop())
print(s.pop())
print(s.is_empty())
```

Output-

```
3
2
2
1
True
PS C:\Users\suruc\PycharmProjects\firstfrog>
```

[Question-6]

main.py > Queue > deQueue

```
class Queue:
    def __init__(self):
        self.s1 = []
        self.s2 = []


    def enQueue(self, x):
        self.s1.append(x)


    def deQueue(self):

        if len(self.s1) == 0 and len(self.s2) == 0:
            print("Q is Empty")
            return

        elif len(self.s2) == 0 and len(self.s1) > 0:
            while len(self.s1):
                temp = self.s1.pop()
                self.s2.append(temp)
            return self.s2.pop()

        else:
            return self.s2.pop()


if __name__ == '__main__':
    q = Queue()
    q.enQueue(1)
    q.enQueue(2)
    q.enQueue(3)

    print(q.deQueue())
    print(q.deQueue())
    print(q.deQueue())
```

Output-

```
1
2
3
PS C:\Users\suruc\PycharmProjects\firstfrog>
```