```
class Stack:
    def __init__(self, capacity=None):
        self.stack = []
        self.capacity = capacity

    def push(self, data):
       if capacity is not None and len(stack) >= capacity:
           print("Stack overflow")
           return None
       self.stack.append(data)

    def pop(self):
        if self.isEmpty():
            print("Stack underflow")
            return None
        return self.stack.pop()

    def peek(self):
        if self.isEmpty():
            print("Stack is empty")
            return None
        return self.stack[-1]

    def isEmpty(self):
        return len(self.stack) == 0

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

    def display(self):
        if self.isEmpty():
            print("Stack is empty.")
        else:
            print("Stack(top->bottom): ")
            for item in self.stack:
                print(item)
```

```
class Queue:
    def __init__(self, capacity=None):
        self.queue = []
        self.capacity = capacity

    def enqueue(self, data):
        if capacity is not None and len(queue) >= capacity:
            print("Queue overflow")
            return None
        self.queue.append(data)

    def dequeue(self):
        if self.is_empty():
            print("Queue underflow")
            return None
        return self.queue.pop(0)

    def front(self):
        if self.is_empty():
            print("Queue is empty")
            return None
        return self.queue[0]

    def is_empty(self):
        return len(self.queue) == 0

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

    def display(self):
        if self.is_empty():
            print("Queue is empty")
        else:
            print("Queue(front->rear): ")
            for item in self.queue:
                print(item, end=" ")
            print()
```

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

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

    def display(self):
        if not self.head:
            print("List is empty")
            return

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

```
PS C:\Users\msdak\OneDrive\Desktop\SEM7\N
list:  1 -> 2 -> 3 -> 4 -> 5 -> None
inserted 0 at position 4
1 -> 2 -> 3 -> 4 -> 0 -> 5 -> None
inserted 0 at position 1
1 -> 0 -> 2 -> 3 -> 4 -> 0 -> 5 -> None
deleted from position 3
1 -> 0 -> 2 -> 4 -> 0 -> 5 -> None
deleted from position 2
1 -> 0 -> 4 -> 0 -> 5 -> None
```

```
PS C:\Users\msdak\OneDrive\Desktop\SEM7\NOP> python -u "c
insertion at end
1 -> None
1 -> 2 -> None
1 -> 2 -> 3 -> None
1 -> 2 -> 3 -> 4 -> None
1 -> 2 -> 3 -> 4 -> 5 -> None
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> None
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> None
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> None
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> None
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> None
deletion from end
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> None
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> None
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> None
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> None
1 -> 2 -> 3 -> 4 -> 5 -> None
1 -> 2 -> 3 -> 4 -> None
1 -> 2 -> 3 -> None
1 -> 2 -> None
1 -> None
List is empty
```

```
PS C:\Users\msdak\OneDrive\Desktop\SEM7\NOP> python -u "
insertion at front
1 -> None
2 -> 1 -> None
3 -> 2 -> 1 -> None
4 -> 3 -> 2 -> 1 -> None
5 -> 4 -> 3 -> 2 -> 1 -> None
6 -> 5 -> 4 -> 3 -> 2 -> 1 -> None
7 -> 6 -> 5 -> 4 -> 3 -> 2 -> 1 -> None
8 -> 7 -> 6 -> 5 -> 4 -> 3 -> 2 -> 1 -> None
9 -> 8 -> 7 -> 6 -> 5 -> 4 -> 3 -> 2 -> 1 -> None
10 -> 9 -> 8 -> 7 -> 6 -> 5 -> 4 -> 3 -> 2 -> 1 -> None
deletion from front
9 -> 8 -> 7 -> 6 -> 5 -> 4 -> 3 -> 2 -> 1 -> None
8 -> 7 -> 6 -> 5 -> 4 -> 3 -> 2 -> 1 -> None
7 -> 6 -> 5 -> 4 -> 3 -> 2 -> 1 -> None
6 -> 5 -> 4 -> 3 -> 2 -> 1 -> None
5 -> 4 -> 3 -> 2 -> 1 -> None
4 -> 3 -> 2 -> 1 -> None
3 -> 2 -> 1 -> None
2 -> 1 -> None
1 -> None
List is empty
```

```
    def insertBegin(self, data):
        newNode = Node(data)
        newNode.next = self.head
        self.head = newNode

    def insertEnd(self, data):
        newNode = Node(data)
        if not self.head:
            self.head = newNode
            return

        current = self.head
        while current.next:
            current = current.next
        current.next = newNode

    def insertAtPos(self, data, pos):
        if pos < 0:
            print("Invalid position")
            return

        newNode = Node(data)

        if pos == 0:
            self.insertBegin(data)
            return

        current = self.head
        for _ in range(pos - 1):
            if current is None:
                print("Position out of bounds")
                return
            current = current.next

        if current is None:
            print("Position out of bounds")
            return

        newNode.next = current.next
        current.next = newNode
```

```
    def deleteBegin(self):
        if not self.head:
            print("List is empty")
            return

        self.head = self.head.next

    def deleteEnd(self):
        if not self.head:
            print("List is empty")
            return

        if self.head.next is None:
            self.head = None
            return

        current = self.head
        while current.next.next:
            current = current.next

        current.next = None

    def deleteAtPos(self, pos):
        if pos < 0:
            print("Invalid position")
            return

        if not self.head:
            print("List is empty")
            return

        if pos == 0:
            self.deleteBegin()
            return

        current = self.head
        for _ in range(pos - 1):
          if current is None or current.next is None:
                print("Position out of bounds")
                return
          current = current.next

        if current.next is None:
            print("Position out of bounds")
            return

        current.next = current.next.next
```

```
PS C:\Users\msdak\OneDrive\Desk
Queue Operation
Enque in the queue
capacity = 5
pushing 1  Queue:  1
pushing 2  Queue:  1 2
pushing 3  Queue:  1 2 3
pushing 4  Queue:  1 2 3 4
pushing 5  Queue:  1 2 3 4 5
pushing 6  Queue overflow
Queue:  1 2 3 4 5
front: 1 & rear: 5
popping ->  Queue:  2 3 4 5
popping ->  Queue:  3 4 5
front: 3 & rear: 5
popping ->  Queue:  4 5
popping ->  Queue:  5
popping ->  Queue is empty
PS C:\Users\msdak\OneDrive\Desk
```

```
PS C:\Users\msdak\OneDrive\Desk
Stack Operation
Pushing in the stack
capacity = 5
pushing 1  Stack:  1
pushing 2  Stack:  1 2
pushing 3  Stack:  1 2 3
pushing 4  Stack:  1 2 3 4
pushing 5  Stack:  1 2 3 4 5
pushing 6  Stack overflow
Stack:  1 2 3 4 5
TOP: 5
popping ->  Stack:  1 2 3 4
popping ->  Stack:  1 2 3
TOP: 3
popping ->  Stack:  1 2
popping ->  Stack:  1
popping ->  Stack is empty.
PS C:\Users\msdak\OneDrive\Desk
```