**Q1**

def rotLeft(a, d):

n = len(a)

d = d % n # Handle cases where d is greater than the array length

rotated\_array = a[d:] + a[:d]

return rotated\_array

# Sample Input

n, d = map(int, input().split())

a = list(map(int, input().split()))

# Perform left rotations

result = rotLeft(a, d)

# Print the rotated array

print(\*result)

**Q2.**

To implement a queue using two stacks and process the given queries, you can follow the code below in Python

class QueueUsingStacks:

def \_\_init\_\_(self):

self.stack1 = [] # For enqueue operations

self.stack2 = [] # For dequeue operations

def enqueue(self, value):

self.stack1.append(value)

def dequeue(self):

if not self.stack2:

while self.stack1:

self.stack2.append(self.stack1.pop())

if self.stack2:

return self.stack2.pop()

else:

return None # Queue is empty

def peek(self):

if not self.stack2:

while self.stack1:

self.stack2.append(self.stack1.pop())

if self.stack2:

return self.stack2[-1]

else:

return None # Queue is empty

# Process queries

def process\_queries(queries):

queue = QueueUsingStacks()

results = []

for query in queries:

query\_type = query[0]

if query\_type == 1:

queue.enqueue(query[1])

elif query\_type == 2:

queue.dequeue()

elif query\_type == 3:

results.append(queue.peek())

return results

# Sample Input

if \_\_name\_\_ == "\_\_main\_\_":

g = int(input().strip())

queries = []

for \_ in range(g):

query = list(map(int, input().split()))

queries.append(query)

# Process queries and print results

results = process\_queries(queries)

for result in results:

print(result)

**Q3** class SinglyLinkedListNode:

def \_\_init\_\_(self, data=None, next=None):

self.data = data

self.next = next

def insertNodeAtPosition(head, data, position):

new\_node = SinglyLinkedListNode(data)

# If the list is empty or we need to insert at the head

if head is None or position == 0:

new\_node.next = head

return new\_node

current = head

for \_ in range(position - 1):

if current.next is not None:

current = current.next

else:

break

new\_node.next = current.next

current.next = new\_node

return head

def printLinkedList(head):

current = head

while current is not None:

print(current.data, end=" ")

current = current.next

print()

# Example usage:

if \_\_name\_\_ == "\_\_main\_\_":

n = int(input())

elements = [int(input()) for \_ in range(n)]

data = int(input())

position = int(input())

# Create the linked list from the input elements

head = None

for element in reversed(elements):

head = insertNodeAtPosition(head, element, 0)

# Insert the new node at the specified position

head = insertNodeAtPosition(head, data, position)

# Print the updated linked list

printLinkedList(head)

**Q5.**

Given the heights of consecutive buildings, find the largest solid area that can be formed within their boundaries.

def LargestRectangle(n, heights):

# Implementation to find the largest rectangle

Pass

Input:

n: Number of buildings

heights: List of n integers representing building heights

Constraints:

1 ≤ n ≤ 10

1 ≤ height < 10

Sample Input:

python

Copy code

n = 3

heights = [1, 2, 1]

Note:

Replace the # Implementation comment with your code to find the largest rectangle within consecutive building heights.