Slip 1:

Q.1) Write a Python program to sort an array of integers using Bubble Sort. [10]

Program:

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]

# Taking user input
arr = []
num_elements = int(input("Enter the number of elements: "))
for i in range(num_elements):
    element = int(input(f"Enter element {i+1}: "))
    arr.append(element)

print("Original array:", arr)
bubble_sort(arr)
print("Sorted array:", arr)
```

Output:

```
Enter the number of elements: 5
Enter element 1: 7
Enter element 2: 3
Enter element 3: 9
Enter element 4: 1
```

Enter element 5: 2

Original array: [7, 3, 9, 1, 2]

Sorted array: [1, 2, 3, 7, 9]

Q.2) Write a Python program for dynamic implementation Stack. [20]

Program:

```
class Stack:
  def __init__(self):
    self.stack = []

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

  def pop(self):
    if not self.is_empty():
      return self.stack.pop()
    else:
      return None

  def peek(self):
    if not self.is_empty():
      return self.stack[-1]
    else:
      return None

  def is_empty(self):
```

```
        return len(self.stack) == 0

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

# Create a new stack
s = Stack()

# Push some items onto the stack
s.push(1)
s.push(2)
s.push(3)
s.push(4)
s.push(5)

print("Initial Stack:")
print(s.stack)

# Peek at the top item
print("Top item:", s.peek())

# Pop an item from the stack
popped_item = s.pop()
print("Popped item:", popped_item)

print("Stack after pop:")
```

```
print(s.stack)


# Check if the stack is empty
print("Is stack empty?", s.is_empty())


# Get the size of the stack
print("Stack size:", s.size())
```

Output:

```
Initial Stack:
[1, 2, 3, 4, 5]
Top item: 5
Popped item: 5
Stack after pop:
[1, 2, 3, 4]
Is stack empty? False
Stack size: 4
```

OR

Q.2) Write a Python program to accept the vertices and edges for a graph and store it as adjacency list and display it. [20]

Program:

```
# Function to create an adjacency list
def create_adjacency_list(vertices, edges):
    # Initialize an empty dictionary for the adjacency list
    adjacency_list = {vertex: [] for vertex in vertices}
    # Add the edges to the adjacency list
```

```
    for edge in edges:
        u, v = edge
        adjacency_list[u].append(v)
        adjacency_list[v].append(u)  # For undirected graphs
    return adjacency_list

# Function to display the adjacency list
def display_adjacency_list(adjacency_list):
    print("Adjacency List of the Graph:")
    for vertex, neighbors in adjacency_list.items():
        print(f"{vertex}: {', '.join(map(str, neighbors))}")

# Main function
if __name__ == "__main__":
    # Accept number of vertices and edges
    num_vertices = int(input("Enter the number of vertices: "))
    num_edges = int(input("Enter the number of edges: "))

    # Accept the vertices
    vertices = []
    print("Enter the vertices:")
    for _ in range(num_vertices):
        vertex = input()
        vertices.append(vertex)

    # Accept the edges
```

```
    edges = []
    print("Enter the edges as pairs of vertices (u v):")
    for _ in range(num_edges):
        u, v = input().split()
        edges.append((u, v))

    # Create the adjacency list
    adjacency_list = create_adjacency_list(vertices, edges)

    # Display the adjacency list
    display_adjacency_list(adjacency_list)
```

**********OUTPUT**********

```
Enter the number of vertices: 4
Enter the number of edges: 4
Enter the vertices:
a
b
c
d
Enter the edges as pairs of vertices (u v):
a b
a c
b d
c d
Adjacency List of the Graph:
```

a: b, c

b: a, d

c: a, d

d: b, c

Slip 2:

Q.1) Write a Python program to sort n elements using Selection Sort. [10]

Program:

```
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_index = i
        for j in range(i + 1, n):
            if arr[j] < arr[min_index]:
                min_index = j
        arr[i], arr[min_index] = arr[min_index], arr[i]

# Get the number of elements from the user
n = int(input("Enter the number of elements: "))

# Get the elements from the user
arr = []
for _ in range(n):
    element = int(input("Enter element: "))
    arr.append(element)
```

```
# Sort the array using Selection Sort
selection_sort(arr)

# Print the sorted array
print("Sorted array:", arr)
```

Output:

```
Enter the number of elements: 4
Enter element: 8
Enter element: 4
Enter element: 9
Enter element: 1
Sorted array: [1, 4, 8, 9]
```

Q.2) Write a menu driven program in Python for the following: [20]

- **To create doubly linked list.**
- **To delete last node from doubly linked list.**
- **Insert a node by a position in doubly linked list**
- **Display.**

Program:

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

class DoublyLinkedList:
```

```
def __init__(self):
    self.head = None

def create_list(self):
    data = input("Enter the data for the node: ")
    new_node = Node(data)
    if self.head is None:
        self.head = new_node
    else:
        current = self.head
        while current.next:
            current = current.next
        current.next = new_node
        new_node.prev = current

def delete_last_node(self):
    if self.head is None:
        print("List is empty. Cannot delete node.")
        return
    current = self.head
    while current.next:
        current = current.next
    current.prev.next = None

def insert_at_position(self):
    pos = int(input("Enter the position to insert the node: "))
```

```
    data = input("Enter the data for the node: ")
    new_node = Node(data)
    if pos == 1:
        new_node.next = self.head
        self.head.prev = new_node
        self.head = new_node
    else:
        current = self.head
        count = 1
        while current and count < pos - 1:
            current = current.next
            count += 1
        if current:
            new_node.next = current.next
            new_node.prev = current
            current.next = new_node
            if new_node.next:
                new_node.next.prev = new_node
        else:
            print("Position out of range.")

def display(self):
    current = self.head
    while current:
        print(current.data, end=" <-> ")
        current = current.next
```

```
        print("None")


def main():
    dll = DoublyLinkedList()
    while True:
        print("\nMenu:")
        print("1. Create doubly linked list")
        print("2. Delete last node from doubly linked list")
        print("3. Insert a node by position in doubly linked list")
        print("4. Display")
        print("5. Exit")
        choice = int(input("Enter your choice: "))
        if choice == 1:
            dll.create_list()
        elif choice == 2:
            dll.delete_last_node()
        elif choice == 3:
            dll.insert_at_position()
        elif choice == 4:
            dll.display()
        elif choice == 5:
            break
        else:
            print("Invalid choice. Please try again.")

if __name__ == "__main__":
```

```
main()
```

Output:

Menu:

1. Create doubly linked list

2. Delete last node from doubly linked list

3. Insert a node by position in doubly linked list

4. Display

5. Exit

Enter your choice: 1

Enter the data for the node: 1 4 7

Menu:

1. Create doubly linked list

2. Delete last node from doubly linked list

3. Insert a node by position in doubly linked list

4. Display

5. Exit

Enter your choice: 4

1 4 7 <-> None

Menu:

1. Create doubly linked list

2. Delete last node from doubly linked list

3. Insert a node by position in doubly linked list

4. Display

5. Exit

Enter your choice: 3

Enter the position to insert the node: 2

Enter the data for the node: 5

Menu:

1. Create doubly linked list

2. Delete last node from doubly linked list

3. Insert a node by position in doubly linked list

4. Display

5. Exit

Enter your choice: 4

1 4 7 <-> 5 <-> None

Menu:

1. Create doubly linked list

2. Delete last node from doubly linked list

3. Insert a node by position in doubly linked list

4. Display

5. Exit

Enter your choice: 2

Menu:

1. Create doubly linked list

2. Delete last node from doubly linked list

3. Insert a node by position in doubly linked list

4. Display

5. Exit

Enter your choice: 4

1 4 7 <-> None

Menu:

1. Create doubly linked list

2. Delete last node from doubly linked list

3. Insert a node by position in doubly linked list

4. Display

5. Exit

Enter your choice:

OR

Q.2) Write a Python program to accept an infix expression and convert it into postfix form. [20]

Program:

```
def prec(c):`
  if c == '^':
    return 3
  elif c == '/' or c == '*':
    return 2
  elif c == '+' or c == '-':
    return 1
  else:
    return -1
```

```
def associativity(c):
    if c == '^':
        return 'R'
    return 'L'  # Default to left-associative


def infix_to_postfix(s):
    result = []
    stack = []

    for i in range(len(s)):
        c = s[i]

        # If the scanned character is an operand, add it to the output string.
        if ('a' <= c <= 'z') or ('A' <= c <= 'Z') or ('0' <= c <= '9'):
            result.append(c)
        # If the scanned character is an '(', push it to the stack.
        elif c == '(':
            stack.append(c)
        # If the scanned character is an ')', pop and add to the output string from the stack
        # until an '(' is encountered.
        elif c == ')':
            while stack and stack[-1] != '(':
                result.append(stack.pop())
            stack.pop()  # Pop '('
        # If an operator is scanned
```

```
        else:
            while stack and (prec(s[i]) < prec(stack[-1]) or
                             (prec(s[i]) == prec(stack[-1]) and associativity(s[i]) == 'L')):
                result.append(stack.pop())
            stack.append(c)

    # Pop all the remaining elements from the stack
    while stack:
        result.append(stack.pop())

    print(''.join(result))

# Driver code
exp = "a+b*(c^d-e)^(f+g*h)-i"

# Function call
infix_to_postfix(exp)
```

Output:

```
abcd^e-fgh*+^*+i-
```

Slip 3:

Q.1) Write a Python program to sort n numbers using Insertion Sort technique. [10]

Program:

```
def insertion_sort(arr):
```

```
    for i in range(1, len(arr)):
        key = arr[i]
        j = i – 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    return arr

def main():
    n = int(input(“Enter the number of elements: “))
    arr = []
    for i in range(n):
        num = int(input(f”Enter element {i + 1}: “))
        arr.append(num)
    print(“Original array:”, arr)
    sorted_arr = insertion_sort(arr)
    print(“Sorted array:”, sorted_arr)

if __name__ == “__main__”:
    main()
```

output:

```
Enter the number of elements: 5
Enter element 1: 5
Enter element 2: 2
Enter element 3: 8
```

Enter element 4: 3

Enter element 5: 1

Original array: [5, 2, 8, 3, 1]

Sorted array: [1, 2, 3, 5, 8]

Q.2) Write a Python program to perform following operation on singly linked list.

** Create the list.**

** Insert an element by position in the list.**

** Delete first element from singly linked list.**

** Display the list. [20]**

Program:

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

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

    def create_list(self):
        num_elements = int(input("Enter the number of elements: "))
        for i in range(num_elements):
            data = int(input(f"Enter element {i + 1}: "))
            self.insert_at_end(data)
```

```
def insert_at_end(self, data):
    if not self.head:
        self.head = Node(data)
    else:
        current = self.head
        while current.next:
            current = current.next
        current.next = Node(data)

def insert_at_position(self, position, data):
    if position < 1:
        print("Invalid position")
        return
    current = self.head
    current_position = 1
    while current and current_position < position – 1:
        current = current.next
        current_position += 1
    if current:
        new_node = Node(data)
        new_node.next = current.next
        current.next = new_node
    else:
        print("Position out of range")

def delete_first_element(self):
```

```
        if self.head:
            self.head = self.head.next
        else:
            print("List is empty")

    def display_list(self):
        current = self.head
        while current:
            print(current.data, end=" ")
            current = current.next
        print()

def main():
    linked_list = LinkedList()
    print("Creating the list...")
    linked_list.create_list()
    print("Original list:", end=" ")
    linked_list.display_list()

    position = int(input("Enter the position to insert a new element: "))
    data = int(input(f"Enter the element to insert at position {position}: "))
    linked_list.insert_at_position(position, data)
    print("List after inserting element:", end=" ")
    linked_list.display_list()

    print("Deleting the first element...")
```

```
    linked_list.delete_first_element()
    print("List after deleting the first element:", end=" ")
    linked_list.display_list()

if __name__ == "__main__":
    main()
```

output:

```
Creating the list...
Enter the number of elements: 5
Enter element 1: 1
Enter element 2: 2
Enter element 3: 3
Enter element 4: 4
Enter element 5: 5
Original list: 1 2 3 4 5
Enter the position to insert a new element: 3
Enter the element to insert at position 3: 10
List after inserting element: 1 2 10 3 4 5
Deleting the first element...
List after deleting the first element: 2 10 3 4 5
```

Q.2) Write a Python program for the dynamic implementation of Queue

Program:

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

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

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

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

def display_queue(self):
    temp = self.front
    while temp:
        print(temp.data, end=" ")
```

```
            temp = temp.next
        print()


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


def main():
    queue = Queue()
    while True:
        print("1. Enqueue")
        print("2. Dequeue")
        print("3. Display Queue")
        print("4. Exit")
        choice = int(input("Enter your choice: "))
        if choice == 1:
            data = int(input("Enter the element to enqueue: "))
            queue.enqueue(data)
        elif choice == 2:
            print("Dequeued element:", queue.dequeue())
        elif choice == 3:
            print("Queue:", end=" ")
            queue.display_queue()
        elif choice == 4:
            break
```

```
        else:
            print("Invalid choice")

if __name__ == "__main__":
    main()
```

output:

```
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 1
Enter the element to enqueue: 10
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 1
Enter the element to enqueue: 20
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 3
Queue: 10 20
1. Enqueue
2. Dequeue
```

3. Display Queue

4. Exit

Enter your choice: 2

Dequeued element: 10

1. Enqueue

2. Dequeue

3. Display Queue

4. Exit

Enter your choice: 3

Queue: 20

Slip 4:

Q1. Write a Python program to search an element in an array using linear search method. [10]

Program:

```
def linear_search(arr, target):
    for i in range(len(arr)):
        if arr[i] == target:
            return i  # Return the index of the element if found
    return -1  # Return -1 if the element is not found

def main():
    arr = []
    num_elements = int(input("Enter the number of elements: "))
```

```
    for i in range(num_elements):
        data = int(input(f"Enter element {i + 1}: "))
        arr.append(data)

    target = int(input("Enter the element to search: "))
    result = linear_search(arr, target)

    if result != -1:
        print(f"Element {target} found at index {result}")
    else:
        print(f"Element {target} not found in the array")

if __name__ == "__main__":
    main()
```

output:

```
Enter the number of elements: 5
Enter element 1: 10
Enter element 2: 20
Enter element 3: 30
Enter element 4: 40
Enter element 5: 50
Enter the element to search: 30
Element 30 found at index 2
```

Q2. Write a Python program to perform following operations on Stack: [20]

- **Push()**

 Pop()

 IsEmpty()

 IsFull()

Program:

```
class Stack:
    def __init__(self, max_size):
        self.max_size = max_size
        self.top = -1
        self.stack = [None] * max_size

    def is_empty(self):
        return self.top == -1

    def is_full(self):
        return self.top == self.max_size - 1

    def push(self, data):
        if self.is_full():
            print("Stack is full. Cannot push element.")
            return
        self.top += 1
        self.stack[self.top] = data

    def pop(self):
        if self.is_empty():
            print("Stack is empty. Cannot pop element.")
```

```
            return
        data = self.stack[self.top]
        self.top -= 1
        return data

    def display_stack(self):
        if self.is_empty():
            print("Stack is empty.")
            return
        for i in range(self.top, -1, -1):
            print(self.stack[i], end=" ")
        print()

def main():
    stack = Stack(5)

    # Push elements
    stack.push(10)
    stack.push(20)
    stack.push(30)

    # Display stack
    print("Stack:", end=" ")
    stack.display_stack()

    # Pop element
```

```
    print("Popped element:", stack.pop())

    # Display stack
    print("Stack:", end=" ")
    stack.display_stack()

    # Check if stack is empty
    print("Is stack empty?", stack.is_empty())

    # Check if stack is full
    print("Is stack full?", stack.is_full())

if __name__ == "__main__":
    main()
```

output:

Stack: 30 20 10

Popped element: 30

Stack: 20 10

Is stack empty? False

Is stack full? False

Q2. Write a menu driven program in python to perform the following operations on Circular singly linked list: [20]

- **Create**
- **Insert a Node by position in the list.**
- **Delete a node by position form the list.**

 Display the List.

Program:

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

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

    def create(self):
        data = int(input("Enter the data for the node: "))
        new_node = Node(data)
        new_node.next = new_node
        self.head = new_node

    def insert_at_position(self, position):
        data = int(input("Enter the data for the new node: "))
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            new_node.next = new_node
        else:
            temp = self.head
            for _ in range(position - 1):
```

```
            temp = temp.next
        new_node.next = temp.next
        temp.next = new_node

    def delete_at_position(self, position):
        if self.head is None:
            print("List is empty. Cannot delete node.")
            return
        temp = self.head
        for _ in range(position - 1):
            temp = temp.next
        node_to_delete = temp.next
        temp.next = node_to_delete.next
        node_to_delete.next = None
        del node_to_delete

    def display(self):
        if self.head is None:
            print("List is empty.")
            return
        temp = self.head
        while True:
            print(temp.data, end=" ")
            temp = temp.next
            if temp == self.head:
                break
```

```
        print()


def main():
    circular_singly_linked_list = CircularSinglyLinkedList()
    while True:
        print("1. Create")
        print("2. Insert at position")
        print("3. Delete at position")
        print("4. Display")
        print("5. Exit")
        choice = int(input("Enter your choice: "))
        if choice == 1:
            circular_singly_linked_list.create()
        elif choice == 2:
            position = int(input("Enter the position to insert: "))
            circular_singly_linked_list.insert_at_position(position)
        elif choice == 3:
            position = int(input("Enter the position to delete: "))
            circular_singly_linked_list.delete_at_position(position)
        elif choice == 4:
            print("List: ", end="")
            circular_singly_linked_list.display()
        elif choice == 5:
            break
        else:
            print("Invalid choice")
```

```
if __name__ == "__main__":
    main()
```

output:

1. Create

2. Insert at position

3. Delete at position

4. Display

5. Exit

Enter your choice: 1

Enter the data for the node: 10

1. Create

2. Insert at position

3. Delete at position

4. Display

5. Exit

Enter your choice: 2

Enter the position to insert: 1

Enter the data for the new node: 20

1. Create

2. Insert at position

3. Delete at position

4. Display

5. Exit

Enter your choice: 4

List: 20 10

1. Create

2. Insert at position

3. Delete at position

4. Display

5. Exit

Enter your choice: 3

Enter the position to delete: 1

1. Create

2. Insert at position

3. Delete at position

4. Display

5. Exit

Enter your choice: 4

List: 10

Slip 5:

Q.1) Write a Python program for the implementation of Priority Queue. [10]

Program:

```
import heapq
class PriorityQueue:
  def __init__(self):
    # Initialize an empty list to represent the priority queue
    self.queue = []
  def is_empty(self):
    # Check if the priority queue is empty
    return len(self.queue) == 0
```

```
def enqueue(self, item, priority):
    # Push the item into the queue with its priority
    # heapq uses a min-heap, so we push (priority, item)
    heapq.heappush(self.queue, (priority, item))
    print(f"Enqueued: {item} with priority {priority}")
def dequeue(self):
    # Pop the item with the highest priority (lowest priority number)
    if not self.is_empty():
        priority, item = heapq.heappop(self.queue)
        print(f"Dequeued: {item} with priority {priority}")
        return item
    else:
        print("Queue is empty, cannot dequeue!")

def peek(self):
    # Peek at the item with the highest priority without removing it
    if not self.is_empty():
        priority, item = self.queue[0]
        print(f"Peek: {item} with priority {priority}")
        return item
    else:
        print("Queue is empty!")

def display(self):
    # Display the priority queue
    print("Priority Queue (from highest to lowest priority):")
```

```
        for priority, item in sorted(self.queue):
            print(f"Item: {item}, Priority: {priority}")

# Main program
if __name__ == "__main__":
    pq = PriorityQueue()

    # Enqueue elements with their priorities
    pq.enqueue('Task 1', 2)
    pq.enqueue('Task 2', 1)
    pq.enqueue('Task 3', 3)

    # Display the queue
    pq.display()

    # Peek the highest priority element
    pq.peek()

    # Dequeue the highest priority element
    pq.dequeue()

    # Display the queue after dequeue
    pq.display()

    # Dequeue another element
    pq.dequeue()
```

```
# Peek the highest priority element again
pq.peek()

# Display the queue after dequeue
pq.display()

# Dequeue another element
pq.dequeue()

# Try to dequeue from an empty queue
pq.dequeue()
```

********OUTPUT**********

Enqueued: Task 1 with priority 2

Enqueued: Task 2 with priority 1

Enqueued: Task 3 with priority 3

Priority Queue (from highest to lowest priority):

Item: Task 2, Priority: 1

Item: Task 1, Priority: 2

Item: Task 3, Priority: 3

Peek: Task 2 with priority 1

Dequeued: Task 2 with priority 1

Priority Queue (from highest to lowest priority):

Item: Task 1, Priority: 2

Item: Task 3, Priority: 3

Dequeued: Task 1 with priority 2

Peek: Task 3 with priority 3

Priority Queue (from highest to lowest priority):

Item: Task 3, Priority: 3

Dequeued: Task 3 with priority 3

Queue is empty, cannot dequeue!

Q.2) Write a Python program to create and display doubly linked list in reverse order. [20]

Program:

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

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

    def append(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
        else:
            temp = self.head
```

```
            while temp.next:
                temp = temp.next
            temp.next = new_node
            new_node.prev = temp

    def display_reverse(self):
        if self.head is None:
            print("List is empty.")
            return
        temp = self.head
        while temp.next:
            temp = temp.next
        while temp:
            print(temp.data, end=" ")
            temp = temp.prev
        print()

def main():
    doubly_linked_list = DoublyLinkedList()
    doubly_linked_list.append(10)
    doubly_linked_list.append(20)
    doubly_linked_list.append(30)
    doubly_linked_list.append(40)
    doubly_linked_list.append(50)

    print("Doubly Linked List in reverse order:")
```

```
    doubly_linked_list.display_reverse()

if __name__ == "__main__":
    main()
```

output:

```
Doubly Linked List in reverse order:
50 40 30 20 10
```

Q.2) Write a Python program to evaluate postfix expression using stack. [20]

Program:

```
# Function to evaluate a postfix expression
def evaluate_postfix(expression):
    # Stack to store operands
    stack = []

    # Split the input expression by spaces to handle multi-digit operands
    tokens = expression.split()

    # Traverse through each token in the expression
    for token in tokens:
        # If the token is a number (operand), push it to the stack
        if token.isdigit():
            stack.append(int(token))
        # If the token is an operator, pop two operands from stack and apply the operator
        else:
            val2 = stack.pop()  # Second operand
```

```
        val1 = stack.pop()  # First operand

        if token == '+':
            stack.append(val1 + val2)
        elif token == '-':
            stack.append(val1 - val2)
        elif token == '*':
            stack.append(val1 * val2)
        elif token == '/':
            stack.append(int(val1 / val2))  # Integer division to match common postfix evaluation

    # The result will be the last remaining element in the stack
    return stack.pop()

# Main program
if __name__ == "__main__":
    # Example postfix expression (with space between operands and operators)
    expression = input("Enter a postfix expression (with space between operands and operators): ")

    # Evaluate the postfix expression
    result = evaluate_postfix(expression)

    # Display the result
    print(f"Result of the postfix expression '{expression}': {result}")
```

**********Output**********

Enter a postfix expression (with space between operands and operators):2 3 1 * + 9 -

Result of the postfix expression '2 3 1 * + 9 -': -4

Slip 6:

Q1. Write a Python program to search an element using binary search method.

[10]

Program:

```
def binary_search(arr, target):
    low = 0
    high = len(arr) - 1

    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    return -1  # not found

def main():
    arr = [2, 5, 8, 12, 16, 23, 38, 56, 72, 91]
    target = int(input("Enter the element to search: "))
```

```
    result = binary_search(arr, target)

    if result != -1:
        print("Element found at index:", result)
    else:
        print("Element not found in the array")

if __name__ == "__main__":
    main()
```

output:

Enter the element to search: 2

Element found at index: 0

Q2. Write a Python program to reverse a Singly Linked List. [20]

Program:

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

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

    def append(self, data):
        new_node = Node(data)
```

```
        if self.head is None:
            self.head = new_node
        else:
            temp = self.head
            while temp.next:
                temp = temp.next
            temp.next = new_node

    def reverse(self):
        prev = None
        current = self.head
        while current:
            next_node = current.next
            current.next = prev
            prev = current
            current = next_node
        self.head = prev

    def display(self):
        temp = self.head
        while temp:
            print(temp.data, end=" ")
            temp = temp.next
        print()

def main():
```

```
    singly_linked_list = SinglyLinkedList()
    singly_linked_list.append(10)
    singly_linked_list.append(20)
    singly_linked_list.append(30)
    singly_linked_list.append(40)
    singly_linked_list.append(50)

    print("Original Singly Linked List:")
    singly_linked_list.display()

    singly_linked_list.reverse()

    print("Reversed Singly Linked List:")
    singly_linked_list.display()

if __name__ == "__main__":
    main()
```

output:

```
Original Singly Linked List:
10 20 30 40 50
Reversed Singly Linked List:
50 40 30 20 10
```

Q2 Write a Python program to create binary search tree (BST) of integer numbers and display it's in- order traversal, pre-order traversal and post-order traversal.

Program:

```
class Node:
    def __init__(self, key):
        self.left = None
        self.right = None
        self.val = key

# In-order traversal
def inorder(root):
    if root:
        inorder(root.left)
        print(root.val, end=' ')
        inorder(root.right)

# Pre-order traversal
def preorder(root):
    if root:
        print(root.val, end=' ')
        preorder(root.left)
        preorder(root.right)

# Post-order traversal
def postorder(root):
    if root:
        postorder(root.left)
        postorder(root.right)
        print(root.val, end=' ')
```

```
# Insert a node
def insert(root, key):
    if root is None:
        return Node(key)
    else:
        if root.val < key:
            root.right = insert(root.right, key)
        else:
            root.left = insert(root.left, key)
    return root

# Driver code
if __name__ == "__main__":
    root = Node(50)
    root = insert(root, 30)
    root = insert(root, 20)
    root = insert(root, 40)
    root = insert(root, 70)
    root = insert(root, 60)
    root = insert(root, 80)

    print("In-order traversal:")
    inorder(root)
    print("\nPre-order traversal:")
    preorder(root)
```

```
    print("\nPost-order traversal:")
    postorder(root)
```

Output:

In-order traversal:

20 30 40 50 60 70 80

Pre-order traversal:

50 30 20 40 70 60 80

Post-order traversal:

20 40 30 60 80 70 50

Slip 7:

Q.1) Write a Python program to sort n numbers using Quick Sort technique.[10]

Program:

```
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quick_sort(left) + middle + quick_sort(right)

def main():
    n = int(input("Enter the number of elements: "))
    arr = []
    for i in range(n):
```

```
        arr.append(int(input("Enter element {}: ".format(i+1))))
    print("Original array:", arr)
    sorted_arr = quick_sort(arr)
    print("Sorted array:", sorted_arr)

if __name__ == "__main__":
    main()
```

output:

```
Enter the number of elements: 5
Enter element 1: 5
Enter element 2: 2
Enter element 3: 8
Enter element 4: 3
Enter element 5: 1
Original array: [5, 2, 8, 3, 1]
Sorted array: [1, 2, 3, 5, 8]
```

Q.2 Write a Python program for the static implementation linear queue. [20]

Program:

```
class LinearQueue:
    def __init__(self, size):
        self.queue = [None] * size
        self.front = 0
        self.rear = 0
        self.size = size
```

```
def is_empty(self):
    return self.front == self.rear
def is_full(self):
    return (self.rear + 1) % self.size == self.front

def enqueue(self, element):
    if self.is_full():
        print("Queue is full. Cannot enqueue.")
        return
    self.queue[self.rear] = element
    self.rear = (self.rear + 1) % self.size

def dequeue(self):
    if self.is_empty():
        print("Queue is empty. Cannot dequeue.")
        return
    element = self.queue[self.front]
    self.front = (self.front + 1) % self.size
    return element

def display(self):
    if self.is_empty():
        print("Queue is empty.")
        return
    i = self.front
    while i != self.rear:
```

```
            print(self.queue[i], end=" ")
            i = (i + 1) % self.size
        print(self.queue[i])

# Example usage:
queue = LinearQueue(5)
queue.enqueue(10)
queue.enqueue(20)
queue.enqueue(30)
queue.display()  # Output: 10 20 30
print(queue.dequeue())  # Output: 10
queue.display()  # Output: 20 30
```

#OR

```
#Q.2) Write a Python program for the implementation of circular queue
class CircularQueue:
    def __init__(self, size):
        self.queue = [None] * size
        self.front = 0
        self.rear = 0
        self.size = size

    def is_empty(self):
        return self.front == self.rear

    def is_full(self):
```

```
    return (self.rear + 1) % self.size == self.front

def enqueue(self, element):
    if self.is_full():
        print("Queue is full. Cannot enqueue.")
        return
    self.queue[self.rear] = element
    self.rear = (self.rear + 1) % self.size

def dequeue(self):
    if self.is_empty():
        print("Queue is empty. Cannot dequeue.")
        return
    element = self.queue[self.front]
    self.front = (self.front + 1) % self.size
    return element

def display(self):
    if self.is_empty():
        print("Queue is empty.")
        return
    i = self.front
    while i != self.rear:
        print(self.queue[i], end=" ")
        i = (i + 1) % self.size
    print(self.queue[i])
```

```
# Example usage:
queue = CircularQueue(5)

print("Initial Queue:")
queue.display()

queue.enqueue(10)
queue.enqueue(20)
queue.enqueue(30)
print("\nAfter Enqueue:")
queue.display()

print("\nDequeued element:", queue.dequeue())
print("After Dequeue:")
queue.display()

queue.enqueue(40)
queue.enqueue(50)
print("\nAfter Enqueue:")
queue.display()

print("\nDequeued element:", queue.dequeue())
print("After Dequeue:")
queue.display()
```

```
print("\nDequeued element:", queue.dequeue())
print("After Dequeue:")
queue.display()

print("\nDequeued element:", queue.dequeue())
print("After Dequeue:")
queue.display()

print("\nDequeued element:", queue.dequeue())
print("After Dequeue:")
queue.display()
```

OUTPUT:-

```
Initial Queue:
Queue is empty.
After Enqueue:
10 20 30
Dequeued element: 10
After Dequeue:
20 30
After Enqueue:
20 30 40 50
Dequeued element: 20
After Dequeue:
30 40 50
Dequeued element: 30
```

After Dequeue:

40 50

Dequeued element: 40

After Dequeue:

50

Dequeued element: 50

After Dequeue:

Queue is empty.

Slip 8:

Q. 1) Write a Python program to search an element in an integer array using: Linear Search Method.

Program:

```
print(end="enter the size:")
arrsize=int(input())
print("enter"+str(arrsize)+"element")
arr=[]
for i in range (arrsize):
  arr.append(input())
print("enter element to search:")
elem=input()
chk=0
for i in range(arrsize):
  if elem==arr[i]:
    index=i+1
    chk=1
    break
if chk==1:
  print("element found at index no. :" +str(index))
else:
```

```
  print("element doesn't found!")
```

Output:

```
enter the size:4
enter4element
11
65
34
54
enter element to search:
65
element found at index no. :2
```

Q. 2) Write a Python program to evaluate postfix expression using stack.

Program:

```
def evaluate_postfix(expression):
  stack = []
  for char in expression:
    if char.isdigit():
      stack.append(int(char))
    else:
      operand2 = stack.pop()
      operand1 = stack.pop()
      if char == '+':
        stack.append(operand1 + operand2)
```

```
        elif char == '-':
            stack.append(operand1 - operand2)
        elif char == '*':
            stack.append(operand1 * operand2)
        elif char == '/':
            stack.append(operand1 / operand2)
    return stack.pop()
postfix_expr = "231*+9-"
result = evaluate_postfix(postfix_expr)
print(f"The result of the postfix expression '{postfix_expr}' is: {result}")
```

Output:

The result of the postfix expression '231*+9-' is: -4

OR

Q.2) Write a Python menu driven program to implement doubly linked list of integers with following operations:

¬ Create

¬ Insert a Node at the end of the list.

¬ Delete specific element

¬ Display

Program:

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

```
        self.prev = None


class DoublyLinkedList:
    def __init__(self):
        self.head = None
    # Insert at the beginning
    def insert_at_beginning(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
        else:
            new_node.next = self.head
            self.head.prev = new_node
            self.head = new_node
        print(f"Inserted {data} at the beginning")
    # Insert at the end
    def insert_at_end(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
        else:
            temp = self.head
            while temp.next is not None:
                temp = temp.next
            temp.next = new_node
            new_node.prev = temp
```

```
    print(f"Inserted {data} at the end")


# Delete from the beginning
def delete_from_beginning(self):
    if self.head is None:
        print("List is empty, no element to delete")
    else:
        temp = self.head
        if self.head.next is None:
            self.head = None
        else:
            self.head = self.head.next
            self.head.prev = None
        print(f"Deleted {temp.data} from the beginning")
# Delete from the end
def delete_from_end(self):
    if self.head is None:
        print("List is empty, no element to delete")
    else:
        temp = self.head
        if temp.next is None:
            self.head = None
        else:
            while temp.next is not None:
                temp = temp.next
            temp.prev.next = None
```

```
            print(f"Deleted {temp.data} from the end")
    # Display the list forward
    def display_forward(self):
        if self.head is None:
            print("List is empty")
        else:
            temp = self.head
            while temp is not None:
                print(temp.data, end=" ")
                temp = temp.next
            print()

    # Display the list backward
    def display_backward(self):
        if self.head is None:
            print("List is empty")
        else:
            temp = self.head
            while temp.next is not None:
                temp = temp.next
            while temp is not None:
                print(temp.data, end=" ")
                temp = temp.prev
            print()

# Menu-driven program
```

```
def menu():
    dll = DoublyLinkedList()
    while True:
        print("\nDoubly Linked List Menu:")
        print("1. Insert at the beginning")
        print("2. Insert at the end")
        print("3. Delete from the beginning")
        print("4. Delete from the end")
        print("5. Display list forward")
        print("6. Display list backward")
        print("7. Exit")

        choice = int(input("Enter your choice: "))

        if choice == 1:
            data = int(input("Enter the value to insert at the beginning: "))
            dll.insert_at_beginning(data)
        elif choice == 2:
            data = int(input("Enter the value to insert at the end: "))
            dll.insert_at_end(data)
        elif choice == 3:
            dll.delete_from_beginning()
        elif choice == 4:
            dll.delete_from_end()
        elif choice == 5:
            print("List in forward direction: ")
```

```
            dll.display_forward()
        elif choice == 6:
            print("List in backward direction: ")
            dll.display_backward()
        elif choice == 7:
            print("Exiting program...")
            break
        else:
            print("Invalid choice, please try again")

if __name__ == "__main__":
    menu()
```

Output:

Doubly Linked List Menu:

1. Insert at the beginning

2. Insert at the end

3. Delete from the beginning

4. Delete from the end

5. Display list forward

6. Display list backward

7. Exit

Enter your choice: 1

Enter the value to insert at the beginning: 3

Inserted 3 at the beginning

Doubly Linked List Menu:

1. Insert at the beginning

2. Insert at the end

3. Delete from the beginning

4. Delete from the end

5. Display list forward

6. Display list backward

7. Exit

Enter your choice: 5

List in forward direction:

3

Doubly Linked List Menu:

1. Insert at the beginning

2. Insert at the end

3. Delete from the beginning

4. Delete from the end

5. Display list forward

6. Display list backward

7. Exit

Enter your choice:

Slip 9:

Q.1) Write a Python program to reverse a given string using stack.

Program:

```
def createStack():
   stack = []
```

```
    return stack
def size(stack):
    return len(stack)
def isEmpty(stack):
    if size(stack) == 0:
        return true
def push(stack, item):
    stack.append(item)
def pop(stack):
    if isEmpty(stack):
        return
    return stack.pop()
def reverse(string):
    n = len(string)
    stack = createStack()
    for i in range(0, n, 1):
        push(stack, string[i])
    string = " "
    for i in range(0, n, 1):
        string += pop(stack)
    return string
string = "Hello World"
string = reverse(string)

print("Reversed string is " + string)
```

Output:

Reversed string is dlroW olleH

Q.2) Write a menu driven Python program for dynamic implementation of Queue .

♣ -Insert

♣ -Delete

♣ -Display

Program:

```
class Queue:
  def __init__(self):
    self.queue = []
  # Enqueue operation
  def enqueue(self, item):
    self.queue.append(item)
    print(f"Enqueued {item}")

  # Dequeue operation
  def dequeue(self):
    if not self.is_empty():
      item = self.queue.pop(0)
      print(f"Dequeued {item}")
      return item
    else:
      print("Queue is empty, cannot dequeue")

  # Peek (front element)
  def peek(self):
    if not self.is_empty():
      print(f"Front element is {self.queue[0]}")
      return self.queue[0]
    else:
      print("Queue is empty")
```

```
    # Check if queue is empty
    def is_empty(self):
        return len(self.queue) == 0

    # Display the queue
    def display(self):
        if self.is_empty():
            print("Queue is empty")
        else:
            print("Queue elements:", ' '.join(map(str, self.queue)))

# Menu-driven program
def menu():
    q = Queue()
    while True:
        print("\nQueue Menu:")
        print("1. Enqueue")
        print("2. Dequeue")
        print("3. Peek")
        print("4. Check if Queue is empty")
        print("5. Display Queue")
        print("6. Exit")

        choice = int(input("Enter your choice: "))

        if choice == 1:
```

```
            item = int(input("Enter the value to enqueue: "))
            q.enqueue(item)
        elif choice == 2:
            q.dequeue()
        elif choice == 3:
            q.peek()
        elif choice == 4:
            if q.is_empty():
                print("Queue is empty")
            else:
                print("Queue is not empty")
        elif choice == 5:
            q.display()
        elif choice == 6:
            print("Exiting program...")
            break
        else:
            print("Invalid choice, please try again")

if __name__ == "__main__":
    menu()
```

*************OUTPUT************

Queue Menu:

1. Enqueue

2. Dequeue

3. Peek

4. Check if Queue is empty

5. Display Queue

6. Exit

Enter your choice: 1

Enter the value to enqueue: 4

Enqueued 4

Queue Menu:

1. Enqueue

2. Dequeue

3. Peek

4. Check if Queue is empty

5. Display Queue

6. Exit

Enter your choice: 1

Enter the value to enqueue: 6

Enqueued 6

Queue Menu:

1. Enqueue

2. Dequeue

3. Peek

4. Check if Queue is empty

5. Display Queue

6. Exit

Enter your choice: 5

Queue elements: 4 6

Queue Menu:

1. Enqueue

2. Dequeue

3. Peek

4. Check if Queue is empty

5. Display Queue

6. Exit

Enter your choice: 2

Dequeued 4

Queue Menu:

1. Enqueue

2. Dequeue

3. Peek

4. Check if Queue is empty

5. Display Queue

6. Exit

Enter your choice: 4

Queue is not empty

Queue Menu:

1. Enqueue

2. Dequeue

3. Peek

4. Check if Queue is empty

5. Display Queue

6. Exit

Enter your choice:

OR

Q.2) Write a menu driven Python program for dynamic implementation of Queue.

□ -Insert

□ -Delete

□ -Display

Program:

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


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


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

```
def enqueue(self, value):
    new_node = Node(value)
    if self.rear is None:  # If the queue is empty
        self.front = self.rear = new_node
        return
    self.rear.next = new_node
    self.rear = new_node

def dequeue(self):
    if self.is_empty():
        return None  # Queue is empty
    removed_node = self.front
    self.front = self.front.next
    if self.front is None:  # If the queue becomes empty
        self.rear = None
    return removed_node.data

def display(self):
    if self.is_empty():
        print("Queue is empty.")
        return
    current = self.front
    while current:
        print(current.data, end=" -> ")
        current = current.next
    print("None")
```

```
def menu():
    print("\nQueue Menu")
    print("1. Enqueue")
    print("2. Dequeue")
    print("3. Display Queue")
    print("4. Check if Queue is Empty")
    print("5. Exit")

def main():
    queue = Queue()
    while True:
        menu()
        choice = int(input("Enter your choice: "))

        if choice == 1:
            value = input("Enter the value to enqueue: ")
            queue.enqueue(value)
            print(f"Enqueued: {value}")

        elif choice == 2:
            dequeued_value = queue.dequeue()
            if dequeued_value is not None:
                print(f"Dequeued: {dequeued_value}")
            else:
                print("Queue is empty. Cannot dequeue.")
```

```
        elif choice == 3:
            print("Current Queue:")
            queue.display()

        elif choice == 4:
            if queue.is_empty():
                print("Queue is empty.")
            else:
                print("Queue is not empty.")

        elif choice == 5:
            print("Exiting program.")
            break

        else:
            print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main()
```

Output:

Queue Menu

1. Enqueue

2. Dequeue

3. Display Queue

4. Check if Queue is Empty

5. Exit

Enter your choice: 1

Enter the value to enqueue: 2

Enqueued: 2

Queue Menu

1. Enqueue

2. Dequeue

3. Display Queue

4. Check if Queue is Empty

5. Exit

Enter your choice: 5

Exiting program.

Slip 10:

Q.1) Write a Python program to search an element in an integer array using: Binary Search Method.

Program:

```
def binary_search(arr, target):
  left, right = 0, len(arr) - 1

  while left <= right:
    mid = left + (right - left) // 2  # Calculate the mid index

    # Check if the target is at mid
    if arr[mid] == target:
```

```
            return mid  # Target found, return the index
        # If target is greater, ignore the left half
        elif arr[mid] < target:
            left = mid + 1
        # If target is smaller, ignore the right half
        else:
            right = mid - 1

    return -1  # Target not found



# Example usage
if __name__ == "__main__":
    # Sorted integer array
    array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    target_value = 6

    result = binary_search(array, target_value)

    if result != -1:
        print(f"Element {target_value} found at index {result}.")
    else:
        print(f"Element {target_value} not found in the array.")
```

Output:

Element 6 found at index 5

Q.2) Write a menu driven program using Python for implementation of doubly

linked list. Menu should have the following options –

□ Create.

□ Display in reverse order.

□ Delete a node at given position.

Program:

```
class CircularQueue:
  def __init__(self, size):
    self.size = size
    self.queue = [None] * size
    self.front = self.rear = -1

  def is_full(self):
    return (self.rear + 1) % self.size == self.front

  def is_empty(self):
    return self.front == -1

  def enqueue(self, item):
    if self.is_full():
      print("Queue is full!")
    else:
      if self.is_empty():
        self.front = self.rear = 0
      else:
        self.rear = (self.rear + 1) % self.size
      self.queue[self.rear] = item
```

```
        print(f"Inserted: {item}")

    def dequeue(self):
        if self.is_empty():
            print("Queue is empty!")
        else:
            removed_item = self.queue[self.front]
            if self.front == self.rear:
                self.front = self.rear = -1  # Queue is now empty
            else:
                self.front = (self.front + 1) % self.size
            return removed_item

    def display(self):
        if self.is_empty():
            print("Queue is empty!")
            return

        if self.rear >= self.front:
            for i in range(self.front, self.rear + 1):
                print(self.queue[i], end=" ")
        else:
            for i in range(self.front, self.size):
                print(self.queue[i], end=" ")
            for i in range(0, self.rear + 1):
                print(self.queue[i], end=" ")
```

```
    print()

def display_reverse(self):
    if self.is_empty():
        print("Queue is empty!")
        return

    reverse = []
    index = self.front

    while True:
        reverse.append(self.queue[index])
        if index == self.rear:
            break
        index = (index + 1) % self.size

    for item in reversed(reverse):
        print(item, end=" ")
    print()

def delete_at_position(self, position):
    if self.is_empty():
        print("Queue is empty!")
        return

    if position < 0 or position >= self.size:
```

```
            print("Invalid position!")
            return

        if position >= self.front and position <= self.rear:
            # Valid deletion position within the existing elements
            for i in range(position, self.rear):
                self.queue[i] = self.queue[(i + 1) % self.size]
            self.queue[self.rear] = None  # Remove the last element
            self.rear = (self.rear - 1) % self.size

            if self.front == self.rear:
                self.front = self.rear = -1  # Queue became empty
        elif position == self.front:
            self.dequeue()  # Just dequeue front
        else:
            print("Position is not valid for deletion!")

# Example usage
if __name__ == "__main__":
    queue = CircularQueue(5)

    queue.enqueue(10)
    queue.enqueue(20)
    queue.enqueue(30)
    queue.enqueue(40)
    queue.enqueue(50)
```

```
print("Current queue:")
queue.display()

print("Queue in reverse order:")
queue.display_reverse()

# Deleting at a given position, e.g., position 1 (which is the second element: 20)
queue.delete_at_position(1)

print("Queue after deletion at position 1:")
queue.display()

print("Queue in reverse order after deletion:")
queue.display_reverse()
```

Output:

Inserted: 10

Inserted: 20

Inserted: 30

Inserted: 40

Inserted: 50

Current queue:

10 20 30 40 50

Queue in reverse order:

50 40 30 20 10

Queue after deletion at position 1:

10 30 40 50

Queue in reverse order after deletion:

50 40 30 10

OR

Q.2) Write a python program for the implementation of a circular queue.

Program:

```
class CircularQueue:
  def __init__(self, size):
    self.size = size
    self.queue = [None] * size
    self.front = -1
    self.rear = -1

  def is_full(self):
    return (self.rear + 1) % self.size == self.front

  def is_empty(self):
    return self.front == -1

  def enqueue(self, item):
    if self.is_full():
      print("Queue is full!")
      return
    if self.front == -1:
      self.front = 0
    self.rear = (self.rear + 1) % self.size
    self.queue[self.rear] = item
    print(f"Enqueued: {item}")

  def dequeue(self):
    if self.is_empty():
      print("Queue is empty!")
      return None
    item = self.queue[self.front]
    if self.front == self.rear:
```

```
            self.front = -1
            self.rear = -1
        else:
            self.front = (self.front + 1) % self.size
        print(f"Dequeued: {item}")
        return item

    def display(self):
        if self.is_empty():
            print("Queue is empty!")
            return
        i = self.front
        print("Queue elements:", end=' ')
        while True:
            print(self.queue[i], end=' ')
            if i == self.rear:
                break
            i = (i + 1) % self.size
        print()


size = int(input("Enter the size of the circular queue: "))
cq = CircularQueue(size)

while True:
    print("\nMenu:")
    print("1. Enqueue")
    print("2. Dequeue")
    print("3. Display")
    print("4. Exit")
    choice = int(input("Enter your choice: "))

    if choice == 1:
        item = int(input("Enter an integer to enqueue: "))
        cq.enqueue(item)
    elif choice == 2:
        cq.dequeue()
    elif choice == 3:
```

```
        cq.display()
    elif choice == 4:
        break
    else:
        print("Invalid choice.")
```

Output:

Enter the size of the circular queue: 5

Menu:

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your choice: 1

Enter an integer to enqueue: 2

Enqueued: 2

Menu:

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your choice: 4

Slip 11:

Q.1) Write a Python program to sort n elements using Selection Sort.

Program:

```
def selectionSort(array, size):
  for ind in range(size):
    min_index = ind
    for j in range(ind + 1, size):
      if array[j] < array[min_index]:
        min_index = j
    (array[ind], array[min_index]) = (array[min_index], array[ind])
  return array
print(end="enter the size: ")
size=int(input())
print("enter" +str(size)+ "element")
array=[]
for i in range (size):
  array.append(input())
print("array before sorting:",array)
array=selectionSort(array,size)
print("array after sorting",selectionSort(array,size))
```

Output:

enter the size: 5

enter5element

32

44

23

43

12

array before sorting: ['32', '44', '23', '43', '12']

array after sorting ['12', '23', '32', '43', '44']

Q.2) Write a Python program for the dynamic implementation stack with following operations: [20]

¬ Push()

¬ Pop()

¬ IsEmpty()

Program:

```
class Stack:
  def __init__(self):  # Fixed the typo here: __inti__ to __init__
    self.stack = []

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

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

  def pop(self):
    if not self.is_empty():
      return self.stack.pop()
    else:
      raise IndexError("Stack is empty. Cannot pop.")  # Fixed: Indexerror to IndexError

  def peek(self):
    if not self.is_empty():
      return self.stack[-1]
    else:
      raise IndexError("Stack is empty. Cannot peek")  # Fixed: Indexerror to IndexError
```

```
# Example usage
stack = Stack()
stack.push(10)
stack.push(20)
stack.push(30)

print("Top element:", stack.peek())
print("Popped element:", stack.pop())
print("Popped element:", stack.pop())
print("Is the stack empty?", stack.is_empty())

stack.push(40)
print("Popped element:", stack.pop())
```

Output:

```
Top element: 30
Popped element: 30
Popped element: 20
Is the stack empty? False
Popped element: 40
```

OR

Q.2) Write a python program to create a Binary Search Tree for integers and display its in-order and post order traversal.

Program:

```
class Node:
    def __init__(self,key):
        self.left=None
        self.right=None
        self.val= key
def insert (root,key):
    if root is None:
        return Node(key)
    if root.val == key:
        return root
    if root.val < key:
        root.right= insert (root.right, key)
    else:
        root.left= insert(root.left, key)
    return root
def inorder (root):
    if root:
        inorder(root.left)
        print(root.val, end=" ")
        inorder(root.right)
def preorder (root):
    if root:
        print(root.val, end=" ")
        preorder(root.left)
        inorder(root.right)
```

```
def postorder (root):
   if root:
      postorder(root.left)
      inorder(root.right)
      print(root.val, end=" ")
r=Node(50)
r=insert(r, 30)
r=insert(r, 20)
r=insert(r, 40)
r=insert(r, 70)
r=insert(r, 60)
r=insert(r, 80)
print(inorder(r))
print(preorder(r))
print(postorder(r))
```

Output:

```
20 30 40 50 60 70 80 None
50 30 20 40 60 70 80 None
20 40 30 60 70 80 50 None
```

Slip 12:

Q.1) Write a Python program to search an element using Linear Search method.

Program:

```
# Function to perform linear search
def linear_search(arr, target):
   for i in range(len(arr)):
```

```
        if arr[i] == target:
            return i  # Return index if found
    return -1  # Return -1 if not found

# Example usage
numbers = [10, 20, 30, 40, 50]  # List of numbers
element_to_find = 30  # Element we want to find

# Call the linear search function
index = linear_search(numbers, element_to_find)

# Print the result
if index != -1:
    print(f"Element {element_to_find} found at index: {index}")
else:
    print(f"Element {element_to_find} not found in the list.")
```

Output:

```
Element 30 found at index: 2
```

Q.2) Write a Python program for Dynamic implementation of Queue with operations:

- **Insert()**
- **Delete()**
- **Display()**

Program:

```
class Queue:
    def __init__(self):
```

```
        self.queue = []

    def insert(self, item):
        self.queue.append(item)  # Add item to the end of the queue
        print(f"Inserted: {item}")

    def delete(self):
        if self.is_empty():
            print("Queue is empty. Cannot delete.")
            return None
        removed_item = self.queue.pop(0)  # Remove item from the front of the queue
        print(f"Deleted: {removed_item}")
        return removed_item

    def display(self):
        if self.is_empty():
            print("Queue is empty.")
        else:
            print("Queue elements:", ' '.join(map(str, self.queue)))

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

# Example usage
if __name__ == "__main__":
    q = Queue()
```

```
q.insert(10)
q.insert(20)
q.insert(30)

q.display()  # Display current elements in the queue

q.delete()  # Delete an element from the queue
q.display()  # Display current elements in the queue

q.delete()  # Delete another element
q.display()  # Display current elements in the queue

q.delete()  # Delete last element
q.display()  # Try to display when the queue is empty
```

Output:

```
Inserted: 10
Inserted: 20
Inserted: 30
Queue elements: 10 20 30
Deleted: 10
Queue elements: 20 30
Deleted: 20
Queue elements: 30
Deleted: 30
Queue is empty.
```

OR

2) Write a Python program to sort an integer array using Quick Sort.

Program:

```
def quick_sort(arr):
    # If the array is empty or has only one element, it's already sorted
    if len(arr) <= 1:
        return arr

    # Choose a pivot (you can choose any element, here we choose the last element)
    pivot = arr[-1]
    left = []
    right = []

    # Partitioning the array into left and right based on the pivot
    for i in range(len(arr) - 1):
        if arr[i] < pivot:
            left.append(arr[i])
        else:
            right.append(arr[i])

    # Recursively apply quick sort to left and right and concatenate the results with the pivot
    return quick_sort(left) + [pivot] + quick_sort(right)

# Example usage
if __name__ == "__main__":
```

```
    array = [10, 7, 8, 9, 1, 5]
    sorted_array = quick_sort(array)
    print("Sorted array:", sorted_array)
```

Output:

Sorted array: [1, 5, 7, 8, 9, 10]

Slip 13:

Q.1) Write a Python program to sort n elements using Merge Sort.

Program:

```
def merge_sort(arr):
    # Base case: if the array is of length 0 or 1, it is already sorted
    if len(arr) <= 1:
        return arr

    # Finding the midpoint of the array
    mid = len(arr) // 2

    # Recursively split the array into two halves
    left_half = merge_sort(arr[:mid])
    right_half = merge_sort(arr[mid:])

    # Merge the sorted halves
    return merge(left_half, right_half)

def merge(left, right):
    sorted_array = []
    i = j = 0
```

```
    # Merge the two halves in a sorted manner
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            sorted_array.append(left[i])
            i += 1
        else:
            sorted_array.append(right[j])
            j += 1

    # If any elements are left in left_half
    while i < len(left):
        sorted_array.append(left[i])
        i += 1

    # If any elements are left in right_half
    while j < len(right):
        sorted_array.append(right[j])
        j += 1

    return sorted_array

# Example usage
if __name__ == "__main__":
    array = [12, 11, 13, 5, 6, 7]
    sorted_array = merge_sort(array)
```

```
  print("Sorted array:", sorted_array)
```

Output:

Sorted array: [5, 6, 7, 11, 12, 13]

Q.2) Write a Python program to implement Linear Queue with operations: [20]

- **Insert()**
- **Delete()**
- **Empty()**

Program:

```
class LinearQueue:
  def __init__(self, size):
    self.size = size  # Maximum size of the queue
    self.queue = []   # List to store queue elements
    self.front = 0    # Points to the front of the queue
    self.rear = -1    # Points to the rear of the queue

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

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

  def insert(self, item):
    if self.is_full():
      print("Queue is full! Cannot insert.")
      return
    self.rear += 1
```

```
        self.queue.append(item)
        print(f"Inserted: {item}")

    def delete(self):
        if self.is_empty():
            print("Queue is empty! Cannot delete.")
            return
        item = self.queue.pop(0)  # Remove the item from the front of the queue
        print(f"Deleted: {item}")
        return item

    def empty(self):
        return self.is_empty()

    def display(self):
        if self.is_empty():
            print("Queue is empty!")
        else:
            print("Queue elements:", self.queue)


# Example usage
if __name__ == "__main__":
    queue_size = 5
    queue = LinearQueue(queue_size)
```

```
queue.insert(10)
queue.insert(20)
queue.insert(30)
queue.display()

queue.delete()
queue.display()

print("Is the queue empty?", queue.empty())

queue.delete()
queue.delete()
queue.delete()  # Trying to delete from an empty queue
```

Output:

Inserted: 10

Inserted: 20

Inserted: 30

Queue elements: [10, 20, 30]

Deleted: 10

Queue elements: [20, 30]

Is the queue empty? False

Deleted: 20

Deleted: 30

Queue is empty! Cannot delete.

OR

Q.2) Write a menu driven program using Python for implementation of singly

linked list. Menu should have the following options – [20]

1. Create.

2. Display.

3. Search specific element in list and display appropriate Message.

4. Delete specific element

Program:

```
class Node:
    def __init__(self, data):
        self.data = data  # Node data
        self.next = None  # Pointer to next node

class SinglyLinkedList:
    def __init__(self):
        self.head = None  # Initialize the head to None

    def create(self, data):
        new_node = Node(data)  # Create a new node
        if self.head is None:
            self.head = new_node  # If the list is empty, set new node as head
        else:
            current = self.head
            while current.next:
                current = current.next  # Move to the end of the list
            current.next = new_node  # Link the new node

    def display(self):
```

```
    if self.head is None:
        print("The list is empty.")
        return
    current = self.head
    while current:
        print(current.data, end=' -> ')
        current = current.next
    print("None")  # End of the linked list

def search(self, key):
    current = self.head
    while current:
        if current.data == key:
            print(f"Element {key} found in the list.")
            return
        current = current.next
    print(f"Element {key} not found in the list.")

def delete(self, key):
    current = self.head
    previous = None
    while current and current.data != key:
        previous = current
        current = current.next

    if current is None:  # Key not found
```

```
            print(f"Element {key} not found in the list.")
            return

        if previous is None:  # Key is at the head
            self.head = current.next
        else:
            previous.next = current.next  # Bypass the current node
        print(f"Element {key} deleted from the list.")

def main():
    linked_list = SinglyLinkedList()  # Create an instance of the list

    while True:
        print("\nMenu:")
        print("1. Create")
        print("2. Display")
        print("3. Search specific element in list")
        print("4. Delete specific element")
        print("5. Exit")

        choice = input("Enter your choice: ")

        if choice == '1':
            data = input("Enter element to insert: ")
            linked_list.create(data)
            print(f"Element {data} added to the list.")
```

```
        elif choice == '2':
            print("Linked List:")
            linked_list.display()

        elif choice == '3':
            key = input("Enter element to search: ")
            linked_list.search(key)

        elif choice == '4':
            key = input("Enter element to delete: ")
            linked_list.delete(key)

        elif choice == '5':
            print("Exiting the program.")
            break

        else:
            print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main()
```

Output:

Menu:

1. Create

2. Display

3. Search specific element in list

4. Delete specific element

5. Exit

Enter your choice: 1

Enter element to insert: 12

Element 12 added to the list.

Menu:

1. Create

2. Display

3. Search specific element in list

4. Delete specific element

5. Exit

Enter your choice: 2

Linked List:

12 -> None

Menu:

1. Create

2. Display

3. Search specific element in list

4. Delete specific element

5. Exit

Enter your choice: 5

Exiting the program.

Slip 14:

Q.1) Write a Python program to sort an element in an integer array using Bubble Sort.

Program:

```
def bubble_sort(arr):
    n = len(arr)
    # Traverse through all elements in the array
    for i in range(n):
        # Last i elements are already sorted
        for j in range(0, n-i-1):
            # Swap if the element found is greater than the next element
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]

# Example usage
if __name__ == "__main__":
    arr = [64, 34, 25, 12, 22, 11, 90]  # Sample array
    print("Original array:", arr)
    bubble_sort(arr)  # Sort the array
    print("Sorted array:", arr)
```

Output:

```
Original array: [64, 34, 25, 12, 22, 11, 90]
Sorted array: [11, 12, 22, 25, 34, 64, 90]
```

Q.2) Write a Python program to check whether given string is palindrome or not using stack.

Program:

```
class Stack:
    def __init__(self):
        self.items = []

    def is_empty(self):
        return self.items == []

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

    def pop(self):
        if not self.is_empty():
            return self.items.pop()

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


def is_palindrome(string):
    stack = Stack()
    cleaned_string = ''.join(char.lower() for char in string if char.isalnum())

    for char in cleaned_string:
        stack.push(char)
```

```
    reversed_string = ''
    while not stack.is_empty():
        reversed_string += stack.pop()

    return cleaned_string == reversed_string

input_string = input("Enter a string: ")
if is_palindrome(input_string):
    print(f"'{input_string}' is a palindrome.")
else:
    print(f"'{input_string}' is not a palindrome.")
```

Output:

```
Enter a string: cyber
'cyber' is not a palindrome.
```

Q.2 Write a Python program for dynamic implementation of stack for integer with Operations: [20]

Program:

- **Push()**
- **Pop()**
- **IsEmpty()**

Program:

```
class Stack:
    def __init__(self):
        self.stack = []  # Initialize an empty stack
```

```
    def push(self, item):
        self.stack.append(item)  # Add item to the top of the stack
        print(f"Pushed {item} to stack.")

    def pop(self):
        if self.is_empty():
            print("Stack is empty. Cannot pop.")
            return None
        item = self.stack.pop()  # Remove and return the top item
        print(f"Popped {item} from stack.")
        return item

    def is_empty(self):
        return len(self.stack) == 0  # Check if the stack is empty


# Example usage
if __name__ == "__main__":
    stack = Stack()  # Create a new stack

    # Push elements to the stack
    stack.push(10)
    stack.push(20)
    stack.push(30)

    # Pop elements from the stack
```

```
    stack.pop()
    stack.pop()

    # Check if the stack is empty
    if stack.is_empty():
        print("Stack is empty.")
    else:
        print("Stack is not empty.")
```

Output:

```
Pushed 10 to stack.
Pushed 20 to stack.
Pushed 30 to stack.
Popped 30 from stack.
Popped 20 from stack.
Stack is not empty.
```

#Slip 15:

#Q.1) Write a Python program to sort n numbers using insertion sort.[10]

```
def insertion_sort (arr):
    n=len(arr)
    if n<=1:
        returrn
    for i in range (1,n):
        key=arr[i]
        j=i-1
```

```
      while  j>=0 and key<arr[j]:
         arr[j+1]=arr[j]
         j-=1
         arr[j+1]=key
   return arr
print(end="enter the size:")
arrsize=int(input())
print("enter" +str(arrsize)+ "element:")
arr=[]
for i in range (arrsize):
   arr.append(input())
print ("before sorting:",arr)
insertion_sort(arr)
print("after sorting:",arr)
```

OUTPUT:-

```
enter the size:5
enter5element:
87
67
45
97
23
before sorting: ['87', '67', '45', '97', '23']
after sorting: ['23', '45', '67', '87', '97']
```

#Q.2) Write a Python program to implement Dynamic Implementation of Queue with following operations:[20]

 # Insert

 # Delete

 # Empty

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


class Queue:
    def _init_(self):
        self.front = None
        self.rear = None

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

    def insert(self, item):
        new_node = Node(item)
        if self.rear is None:
            self.front = self.rear = new_node
        else:
```

```
        self.rear.next = new_node
        self.rear = new_node
    print(f"Inserted: {item}")

def delete(self):
    if self.is_empty():
        print("Queue is empty. Cannot delete item.")
        return None
    removed_data = self.front.data
    self.front = self.front.next
    if self.front is None:
        self.rear = None
    print(f"Deleted: {removed_data}")
    return removed_data

def display(self):
    if self.is_empty():
        print("Queue is empty.")
        return
    current = self.front
    print("Current Queue: ", end="")
    while current:
        print(current.data, end=" ")
        current = current.next
    print()
```

```
if _name_ == "_main_":
    queue = Queue()
    queue.insert(10)
    queue.insert(20)
    queue.insert(30)
    queue.display()
    queue.delete()
    queue.display()

    if queue.is_empty():
        print("The queue is empty.")
    else:
        print("The queue is not empty.")

    queue.delete()
    queue.delete()
    queue.delete()
    queue.display()

    if queue.is_empty():
        print("The queue is empty.")
```

OUTPUT:-

```
Inserted: 10
Inserted: 20
```

Inserted: 30

Current Queue: 10 20 30

Deleted: 10

Current Queue: 20 30

The queue is not empty.

Deleted: 20

Deleted: 30

Queue is empty. Cannot delete item.

Queue is empty.

The queue is empty.

#OR

#Q.2 Write a python program to create Binary Search Tree for integers and display it's pre-order and post-order traversal. [20]

```
class Node:
    def _init_(self, key):
        self.val = key
        self.left = None
        self.right = None

def insert(root, key):
    if root is None:
        return Node(key)
    if root.val == key:
```

```
        return root
    if root.val < key:
        root.right = insert(root.right, key)
    else:
        root.left = insert(root.left, key)
    return root

def preorder(root):
    if root:
        print(root.val, end=" ")
        preorder(root.left)
        preorder(root.right)

def postorder(root):
    if root:
        postorder(root.left)
        postorder(root.right)
        print(root.val, end=" ")



r = Node(50)
r = insert(r,30)
r = insert(r,20)
r = insert(r,40)
r = insert(r,70)
r = insert(r,60)
```

```
r = insert(r,80)


print("Preorder Traversal:")
preorder(r)
print("\nPostorder Traversal:")
postorder(r)
```

OUTPUT:-

```
Preorder Traversal:
50 30 20 40 70 60 80
Postorder Traversal:
20 40 30 60 80 70 50
```

#Slip 16:

#Q.1) Write a Python program to sort an integer array using a Selection Sort.[10]

```
def selection_sort (arr,arrsize):
   n = len(arr)
   for i in range (arrsize):
      min_index=i
      for j  in range (i+1, arrsize):
         if arr[j]<arr[min_index]:
            min_index=j
            (arr[i],arr[min_index])=(arr[min_index],arr[i])
   return arr
print(end="enter the size:")
```

```
arrsize=int(input())
print("enter "+str(arrsize)+" element:")
arr=[]
for i in range (arrsize):
   arr.append(input())
print("array before sorting", arr)
arr=selection_sort(arr,arrsize)
print("array after sorting",selection_sort(arr,arrsize))
```

OUTPUT:-

```
enter the size:5
enter 5 element:
56
90
78
12
10
array before sorting ['56', '90', '78', '12', '10']
array after sorting ['10', '12', '56', '78', '90']
```

#Q.2) Write a Python program to reverse a singly linked list:[20]

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

class slinkedlist:
    def _init_(self):
        self.head = None

    def listprint(self):
        temp = self.head
        while temp is not None:
            print(temp.data)
            temp = temp.next

    def reverse_list(self):
        if self.head is None:
            return None
        previous = None
        current = self.head
        after = None

        while current:
            after = current.next
            current.next = previous
            previous = current
```

```
        current = after


    self.head = previous



list = slinkedlist()
list.head = Node(10)
e2 = Node(20)
e3 = Node(30)
list.head.next = e2
e2.next = e3

list.listprint()
print("After reversing the list")
list.reverse_list()
list.listprint()
```

OUTPUT:-

```
10
20
30
After reversing the list
30
20
10
```

#OR

#Q.2) Write a Python program to convert infix expression into Postfix. [20]

```
def prec(c):
    if c == '^':
        return 3
    elif c == '/' or c == '*':
        return 2
    elif c == '+' or c == '-':
        return 1
    else:
        return -1

def associativity(c):
    if c == '^':
        return 'R'
    return 'L'

def infix_to_postfix(s):
    result = []
    stack = []

    for i in range(len(s)):
        c = s[i]

        if ('a' <= c <= 'z') or ('A' <= c <= 'Z') or ('0' <= c <= '9'):
```

```
            result.append(c)

        elif c == '(':
            stack.append(c)

        elif c == ')':
            while stack and stack[-1] != '(':
                result.append(stack.pop())
            stack.pop()
        else:
            while stack and (prec(s[i]) < prec(stack[-1]) or
                             (prec(s[i]) == prec(stack[-1]) and associativity(s[i]) == 'L')):
                result.append(stack.pop())
            stack.append(c)


    while stack:
        result.append(stack.pop())

    print(''.join(result))


exp = "a+b-c(d^e/b)*a*c+d"

infix_to_postfix(exp)
```

OUTPUT:-

ab+cde^b/a*c*-d+

#Slip 17:

#Q.1) Write a Python program to accept polynomial and display it. [10]

```
def input_polynomial():
    degree = int(input("Enter the degree of the polynomial: "))
    coefficients = {}
    for i in range(degree, -1, -1):
        coef = float(input(f"Enter the coefficient for x^{i}: "))
        coefficients[i] = coef
    return coefficients

def display_polynomial(coefficients):
    polynomial_str = ""

    for power, coef in sorted(coefficients.items(), reverse=True):
        if coef != 0:
            if power == 0:
                polynomial_str += f"{coef}"
            elif power == 1:
                polynomial_str += f"{coef}x"
            else:
                polynomial_str += f"{coef}x^{power}"
            polynomial_str += " + "
```

```
    if polynomial_str.endswith(" + "):
        polynomial_str = polynomial_str[:-3]

    print("The polynomial is:")
    print(polynomial_str if polynomial_str else "0")
coefficients = input_polynomial()
display_polynomial(coefficients)
```

OUTPUT:-

```
Enter the degree of the polynomial: 4
Enter the coefficient for x^4: 6
Enter the coefficient for x^3: 7
Enter the coefficient for x^2: 3
Enter the coefficient for x^1: 0
Enter the coefficient for x^0: 2
The polynomial is:
6.0x^4 + 7.0x^3 + 3.0x^2 + 2.0
```

#Q.2) Write a Python program to convert infix expression to prefix expression.[20]

```
def is_operator(c):
    return c in ['+', '-', '*', '/', '^']

def precedence(c):
    if c == '+' or c == '-':
        return 1
```

```
    elif c == '*' or c == '/':
        return 2
    elif c == '^':
        return 3
    return -1

def infix_to_prefix(infix_expr):
    infix_expr = infix_expr[::-1]
    infix_expr = list(infix_expr)
    for i in range(len(infix_expr)):
        if infix_expr[i] == '(':
            infix_expr[i] = ')'
        elif infix_expr[i] == ')':
            infix_expr[i] = '('
    infix_expr = ''.join(infix_expr)
    postfix_expr = infix_to_postfix(infix_expr)
    prefix_expr = postfix_expr[::-1]

    return prefix_expr
def infix_to_postfix(infix_expr):
    stack = []
    result = ''

    for c in infix_expr:
        if c.isalpha() or c.isdigit():
            result += c
```

```
        elif c == '(':
            stack.append(c)
        elif c == ')':
            while stack and stack[-1] != '(':
                result += stack.pop()
            stack.pop()
        else:
            while stack and precedence(c) <= precedence(stack[-1]):
                result += stack.pop()
            stack.append(c)

    while stack:
        result += stack.pop()

    return result

infix = "(A-B/C)*(A/K-L)"
prefix = infix_to_prefix(infix)
print(f"Infix: {infix}")
print(f"Prefix: {prefix}")
```

OUTPUT:-

Infix: (A-B/C)*(A/K-L)

Prefix: *-A/BC-/AKL

#OR

#Q. 2) Write a Python program to accept a graph for n vertices and display it by using adjacency matrix. [20]

```
def create_adjacency_matrix(n):
    matrix = [[0 for _ in range(n)] for _ in range(n)]
    edges = int(input(f"Enter the number of edges: "))
    print("Enter the edges in the format: vertex1 vertex2")
    for _ in range(edges):
        u, v = map(int, input().split())
        matrix[u][v] = 1
        matrix[v][u] = 1

    return matrix
def display_adjacency_matrix(matrix):
    print("Adjacency Matrix:")
    for row in matrix:
        print(" ".join(map(str, row)))

if __name__ == "__main__":
    n = int(input("Enter the number of vertices: "))
    adj_matrix = create_adjacency_matrix(n)
    display_adjacency_matrix(adj_matrix)
```

OUTPUT:-

```
Enter the number of vertices: 4
Enter the number of edges: 4
Enter the edges in the format: vertex1 vertex2
```

0 1

0 2

2 3

1 3

Adjacency Matrix:

0 1 1 0

1 0 0 1

1 0 0 1

0 1 1 0

#Slip 18:

#Q.1) Write a Python program to sort an element in an integer array using: Selection Sort [10]

```
def selection_sort (arr,arrsize):
    n = len(arr)
    for ind in range (arrsize):
        min_index=ind
        for j  in range (ind+1, arrsize):
            if arr[j]<arr[min_index]:
                min_index=j
                (arr[ind],arr[min_index])=(arr[min_index],arr[ind])
    return arr
```

```
print(end="enter the size:")
arrsize=int(input())
print("enter "+str(arrsize)+" element:")
arr=[]
for i in range (arrsize):
   arr.append(input())
print("array before sorting", arr)
arr=selection_sort(arr,arrsize)
print("array after sorting",selection_sort(arr,arrsize))
```

OUTPUT:-

```
enter the size:5
enter 5 element:
86
12
10
36
90
array before sorting ['86', '12', '10', '36', '90']
array after sorting ['10', '12', '36', '86', '90']
```

 # Q.2) Write a Python program to sort singly linked list. [20]

```
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
        current = self.head
        while current.next:
            current = current.next
        current.next = new_node
    def print_list(self):
        current = self.head
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")
    def get_middle(self, head):
        if head is None:
            return head
        slow = head
        fast = head
        while fast.next and fast.next.next:
            slow = slow.next
            fast = fast.next.next
```

```
    return slow
  def sorted_merge(self, left, right):
    if left is None:
      return right
    if right is None:
      return left
    if left.data <= right.data:
      result = left
      result.next = self.sorted_merge(left.next, right)
    else:
      result = right
      result.next = self.sorted_merge(left, right.next)

    return result
  def merge_sort(self, head):
    if head is None or head.next is None:
      return head
    middle = self.get_middle(head)
    next_to_middle = middle.next
    middle.next = None
    left = self.merge_sort(head)
    right = self.merge_sort(next_to_middle)
    sorted_list = self.sorted_merge(left, right)

    return sorted_list
  def sort(self):
```

```
        self.head = self.merge_sort(self.head)
if __name__ == "__main__":
    llist = LinkedList()
    elements = list(map(int, input("Enter the elements of the linked list separated by space: ").split()))
    for element in elements:
        llist.append(element)


    print("Original Linked List:")
    llist.print_list()
    llist.sort()
    print("Sorted Linked List:")
    llist.print_list()
```

OUTPUT:-

```
Enter the elements of the linked list separated by space:  2 4 6 8 2 5 7
Original Linked List:
2 -> 4 -> 6 -> 8 -> 2 -> 5 -> 7 -> None
Sorted Linked List:
2 -> 2 -> 4 -> 5 -> 6 -> 7 -> 8 -> None
```

#OR

#Q. 2 Write a menu driven program in Python for the following operations on circular singly linked list. [20]

#1. Create.

#2. Display.

#3. Delete specific element

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

class circularsll:
    def _init_(self):
        self.head = None

    def create(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            new_node.next = self.head
        else:
            temp = self.head
            while temp.next != self.head:
                temp = temp.next
            temp.next = new_node
            new_node.next = self.head

    def display(self):
        if self.head is None:
            print("List is empty.")
            return
        temp = self.head
```

```
    while True:
        print(temp.data, end=" -> ")
        temp = temp.next
        if temp == self.head:
            break
    print("(back to head)")

def delete(self, value):
    if self.head is None:
        print("List is empty. Cannot delete.")
        return

    if self.head.data == value and self.head.next == self.head:
        self.head = None
        return

    current = self.head
    previous = None

    while True:
        if current.data == value:
            if previous is None:
                last_node = self.head
                while last_node.next != self.head:
                    last_node = last_node.next
                last_node.next = current.next
```

```
            self.head = current.next
        else:
            previous.next = current.next
        print(f"Deleted: {value}")
        return
      previous = current
      current = current.next
      if current == self.head:
        break

    print(f"Value {value} not found in the list.")

def main():
  cll = circularsll()

  while True:
    print("\nMenu:")
    print("1. Create Node")
    print("2. Display List")
    print("3. Delete Specific Element")
    print("4. Exit")
    choice = input("Enter your choice: ")

    if choice == '1':
      data = input("Enter data for new node: ")
      cll.create(data)
```

```
        elif choice == '2':
            cll.display()
        elif choice == '3':
            value = input("Enter value to delete: ")
            cll.delete(value)
        elif choice == '4':
            print("Exiting...")
            break
        else:
            print("Invalid choice. Please try again.")

if _name_ == "_main_":
    main()
```

OUTPUT:-

```
Menu:
1. Create Node
2. Display List
3. Delete Specific Element
4. Exit
Enter your choice: 1
Enter data for new node: 33

Menu:
1. Create Node
2. Display List
3. Delete Specific Element
```

4. Exit

Enter your choice: 1

Enter data for new node: 23

Menu:

1. Create Node

2. Display List

3. Delete Specific Element

4. Exit

Enter your choice: 1

Enter data for new node: 45

Menu:

1. Create Node

2. Display List

3. Delete Specific Element

4. Exit

Enter your choice: 1

Enter data for new node: 67

Menu:

1. Create Node

2. Display List

3. Delete Specific Element

4. Exit

Enter your choice: 2

33 -> 23 -> 45 -> 67 -> (back to head)

Menu:

1. Create Node

2. Display List

3. Delete Specific Element

4. Exit

Enter your choice: 3

Enter value to delete: 45

Deleted: 45

Menu:

1. Create Node

2. Display List

3. Delete Specific Element

4. Exit

Enter your choice: 2

33 -> 23 -> 67 -> (back to head)

Menu:

1. Create Node

2. Display List

3. Delete Specific Element

4. Exit

Enter your choice: 4

Exiting...

#Slip 19:

#Q1. Write a Python program to sort the elements in an integer array using: insertion Sort. [10]

```
def insertion_sort (arr):
   n=len(arr)
   if n<=1:
      returrn
   for i in range (1,n):
      key=arr[i]
      j=i-1
      while  j>=0 and key<arr[j]:
         arr[j+1]=arr[j]
         j-=1
         arr[j+1]=key
   return arr
print(end="enter the size:")
arrsize=int(input())
print("enter " +str(arrsize)+ " element:")
arr=[]
for i in range (arrsize):
   arr.append(input())
print ("before sorting:",arr)
insertion_sort(arr)
print("after sorting:",arr)
```

OUTPUT:-

enter the size:5

enter 5 element:

34

12

78

90

97

before sorting: ['34', '12', '78', '90', '97']

after sorting: ['12', '34', '78', '90', '97']

#Q.2 Write a Python program to accept string from user and store its character one by one into the nodes of singly linked list and display that list. [20]

```
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
```

```
        current = self.head
        while current.next:
            current = current.next
        current.next = new_node
    def print_list(self):
        current = self.head
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")
if __name__ == "__main__":
    llist = LinkedList()
    user_string = input("Enter a string: ")
    for char in user_string:
        llist.append(char)
    print("Characters in the linked list:")
    llist.print_list()
```

OUTPUT:-

Enter a string: 678954420

Characters in the linked list:

6 -> 7 -> 8 -> 9 -> 5 -> 4 -> 4 -> 2 -> 0 -> None

#OR

#Q.2) Write a Python program for the evaluation of given postfix expression. [20]

```
def e_postfix(expression):
    stack = []

    operators = set(['+', '-', '*', '/'])

    for char in expression:
        if char.isdigit():
            stack.append(int(char))
        elif char in operators:
            operand2 = stack.pop()
            operand1 = stack.pop()

            if char == '+':
                stack.append(operand1 + operand2)
            elif char == '-':
                stack.append(operand1 - operand2)
            elif char == '*':
                stack.append(operand1 * operand2)
            elif char == '/':
                stack.append(int(operand1 / operand2))
        else:
            raise ValueError("Invalid character in expression: " + char)

    return stack.pop() if stack else None

if _name_ == "_main_":
```

```
postfix_expression = "23*54*+9-"
result = e_postfix(postfix_expression)
print("Result of postfix expression '" + postfix_expression + "': " + str(result))
```

OUTPUT:-

```
Result of postfix expression '23*54*+9-': 17
```

#Slip 20:

#Q 1. Write a Python program to sort the data by using insertion sort technique.[10]

```
def insertion_sort (arr):
    n=len(arr)
    if n<=1:
        returrn
    for i in range (1,n):
        key=arr[i]
        j=i-1
        while  j>=0 and key<arr[j]:
            arr[j+1]=arr[j]
            j-=1
            arr[j+1]=key
    return arr
```

```
print(end="enter the size:")
arrsize=int(input())
print("enter " +str(arrsize)+ " element:")
arr=[]
for i in range (arrsize):
   arr.append(input())
print ("before sorting:",arr)
insertion_sort(arr)
print("after sorting:",arr)
```

OUTPUT:-

```
enter the size:5
enter 5 element:
98
76
56
23
10
before sorting: ['98', '76', '56', '23', '10']
after sorting: ['10', '23', '56', '76', '98']
```

#Q 2. Write a Python program to create doubly linked list for n integers, calculate their sum and display it.[20]

```
class Node:
   def _init_(self, data):
```

```
        self.data = data
        self.next = None
        self.prev = None

class DoublyLinkedList:
    def _init_(self):
        self.head = None

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

    def calculate_sum(self):
        total_sum = 0
        current = self.head
        while current:
            total_sum += current.data
            current = current.next
        return total_sum
```

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

def main():
    n = int(input("Enter the number of integers to be added to the doubly linked list: "))
    dll = DoublyLinkedList()

    for _ in range(n):
        data = int(input("Enter an integer: "))
        dll.append(data)

    print("The doubly linked list is:")
    dll.display()

    total_sum = dll.calculate_sum()
    print(f"The sum of the integers in the doubly linked list is: {total_sum}")

if _name_ == "_main_":
    main()
```

OUTPUT:-

Enter the number of integers to be added to the doubly linked list: 5

Enter an integer: 23

Enter an integer: 45

Enter an integer: 67

Enter an integer: 89

Enter an integer: 20

The doubly linked list is:

23 <-> 45 <-> 67 <-> 89 <-> 20 <-> None

The sum of the integers in the doubly linked list is: 244

#OR

#Q 2. Write a Python program to create a singly linked list and display its alternate nodes.[20]

```
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
```

```
        current = self.head
        while current.next:
            current = current.next
        current.next = new_node
    def print_alternate_nodes(self):
        current = self.head
        index = 0
        while current:
            if index % 2 == 0:
                print(current.data, end=" -> ")
            current = current.next
            index += 1
        print("None")
if __name__ == "__main__":
    llist = LinkedList()
    elements = list(map(int, input("Enter the elements of the linked list separated by space: ").split()))
    for element in elements:
        llist.append(element)
    print("Alternate nodes in the linked list:")
    llist.print_alternate_nodes()
```

OUTPUT:-

```
Enter the elements of the linked list separated by space:  1 4 2 5 3
Alternate nodes in the linked list:
1 -> 2 -> 3 -> None
```