Parul University

FACULTY OF ENGINEERING & TECHNOLOGY

BACHELOR OF TECHNOLOGY

COMETITIVE CODING LABORATORY

(303105259)

4th SEMESTER

COMPUTER SCIENCE & ENGINEERING DEPARTMENT

Laboratory Mannual

INSTRUCTIONS TO STUDENTS

1. Every student should obtain a copy of laboratory Manual.
2. Dress Code: Students must come to the laboratory wearing.
   i. Trousers,
   ii. half-sleeve tops and
   iii. Leather shoes. Half pants, loosely hanging garments and slippers are not allowed.
3. To avoid injury, the student must take the permission of the laboratory staff before handling any machine.
4. Students must ensure that their work areas are clean and dry to avoid slipping.
5. Do not eat or drink in the laboratory.
6. Do not remove anything from the computer laboratory without permission.
7. Do not touch, connect or disconnect any plug or cable without your lecturer/laboratory technician's permission.
8. All students need to perform the practical/program.

CERTIFICATE

This is to certify that
Ms. PEEHU KIRTI *with enrolment no* ***2303031050461*** *has successfully completed his/her laboratory experiments in the* ***Competitive Coding Laboratory (303105259)*** *from the department of* ***COMPUTER SCIENCE AND ENGINEERING*** *during the academic year 2024-25.*

Date of Submission: ..... Staff In charge: .....

Head of Department: .....

Practical-1

AIM: Write a program for implementing a MINSTACK which should support operations like push, pop, overflow, underflow, display
1. Construct a stack of N-capacity 2. Push elements 3. Pop elements
4. Top element 5. Retrieve the min element from the stack

INPUT:

```
class MinStack:
    def __init__(self):
        self.stack = []
        self.min_stack = []
    def push(self, val):
        self.stack.append(val)
        if not self.min_stack or val <= self.min_stack[-1]:
            self.min_stack.append(val)
    def pop(self):
        if not self.stack:
            return "Stack is empty!"
        popped = self.stack.pop()
        if popped == self.min_stack[-1]:
            self.min_stack.pop()
        return popped
    def top(self):
        if not self.stack:
            return "Stack is empty!"
        return self.stack[-1]
    def get_min(self):
        if not self.min_stack:
            return "Stack is empty!"
        return self.min_stack[-1]
min_stack = MinStack()
min_stack.push(5)
min_stack.push(3)
min_stack.push(7)
print("Top Element:", min_stack.top())
print("Minimum Element:", min_stack.get_min())
min_stack.pop()
print("After Pop - Top Element:", min_stack.top())
print("After Pop - Minimum Element:", min_stack.get_min())
```

OUTPUT:

Practical-2

AIM: Write a program to deal with real-world situations where Stack data structure is widely used Evaluation of expression: Stacks are used to evaluate expressions, especially in languages that use postfix or prefix notation. Operators and operands are pushed onto the stack, and operations are performed based on the LIFO principle.

INPUT:

```
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()
if __name__ == "__main__":
    expression = "231*+9-"
    print("Postfix Expression:", expression)
    result = evaluate_postfix(expression)
    print("Result:", result)
```

OUTPUT:

Practical-3

AIM: Write a program for finding NGE NEXT GREATER ELEMENT from an array.

INPUT:

```
def find_next_greater_elements(arr):
    n = len(arr)
    nge_result = []
    for i in range(n):
        nge = -1
        for j in range(i+1, n):
            if arr[j] > arr[i]:
                nge = arr[j]
                break
        nge_result.append((arr[i], nge))
    return nge_result

n = int(input("Enter the number of elements: "))
arr = []
print("Enter the element: ")
for _ in range(n):
    arr.append(int(input()))
nge_list = find_next_greater_elements(arr)
for element, nge in nge_list:
    print(element, "-->", nge)
```

OUTPUT:

Practical-4

AIM: Write a program to design a circular queue(k) which Should implement the below functions
a. Enqueue b. Dequeue c. Front d. Rear

INPUT:

```
class CircularQueue:
    def __init__(self, k):
        self.capacity = k
        self.queue = [-1] * k
        self.front_index = -1
        self.rear_index = -1
        self.size = 0
    def enqueue(self, value):
        if self.size == self.capacity:
            print("Queue is full.")
            return
        if self.size == 0:
            self.front_index = 0
        self.rear_index = (self.rear_index + 1) % self.capacity
        self.queue[self.rear_index] = value
        self.size += 1
    def dequeue(self):
        if self.size == 0:
            print("Queue is empty.")
            return -1
        removed_value = self.queue[self.front_index]
        if self.front_index == self.rear_index:
            self.front_index = -1
            self.rear_index = -1
        else:
            self.front_index = (self.front_index + 1) % self.capacity
        self.size -= 1
        return removed_value
    def front(self):
        if self.size == 0:
            print("Queue is empty.")
            return -1
        return self.queue[self.front_index]
    def rear(self):
        if self.size == 0:
            print("Queue is empty.")
            return -1
        return self.queue[self.rear_index]
if __name__ == "__main__":
```

```
queue = CircularQueue(5)
queue.enqueue(1)
queue.enqueue(2)
queue.enqueue(3)
queue.enqueue(4)
queue.enqueue(5)
print("Front:", queue.front())
print("Rear:", queue.rear())
print("Dequeuing:", queue.dequeue())
print("Dequeuing:", queue.dequeue())
print("Front:", queue.front())
print("Rear:", queue.rear())
queue.enqueue(6)
queue.enqueue(7)
print("Front:", queue.front())
print("Rear:", queue.rear())
```

OUTPUT:

Practical-5

AIM: Write a Program for an infix expression, and convert it to postfix notation. Use a queue to implement the Shunting Yard Algorithm for expression conversion.

INPUT:

```
from queue import Queue
def precedence(op):
    if op in ('+', '-'):
        return 1
    if op in ('*', '/'):
        return 2
    return 0
def is_operator(char):
    return char in ('+', '-', '*', '/')
def infix_to_postfix(expression):
    output = Queue()
    operators = []
    for char in expression:
        if char.isalnum():
            output.put(char)
        elif char == '(':
            operators.append(char)
        elif char == ')':
            while operators and operators[-1] != '(':
                output.put(operators.pop())
            operators.pop()
        elif is_operator(char):
            while (operators and operators[-1] != '(' and
                   precedence(operators[-1]) >= precedence(char)):
                output.put(operators.pop())
            operators.append(char)
    while operators:
        output.put(operators.pop())
    postfix = []
    while not output.empty():
        postfix.append(output.get())
    return ''.join(postfix)
if __name__ == "__main__":
    infix_expression = "A+B*(C-D)"
    print("Infix Expression:", infix_expression)
    postfix_expression = infix_to_postfix(infix_expression)
    print("Postfix Expression:", postfix_expression)
```

OUTPUT: