Python Programming and Data Structure Programs (20 Marks programs)

1. Write a Python program to perform infix to postfix conversion of given expression using stack.(A+(C+D*F+T*A)*B/C]

Ans:

```
# Function to check if a given character is an operator

def is_operator(char):
    operators = ['+', '-', '*', '/', '^']
    return char in operators

# Function to get the precedence of an operator
def precedence(operator):
    if operator == '+' or operator == '-':
        return 1
    elif operator == '*' or operator == '/':
        return 2
    elif operator == '^':
        return 3
    else:
        return -1

# Function to convert infix to postfix
def infix_to_postfix(expression):
    postfix = ""
    stack = []
    for char in expression:
        # If the character is an operand, add it to the postfix string
        if char.isalnum():
            postfix += char
        # If the character is an opening parenthesis, push it to the stack
        elif char == '(':
            stack.append('(')
        # If the character is a closing parenthesis, pop operators from the
stack and add them to the postfix string until an opening parenthesis is
encountered
        elif char == ')':
            while stack and stack[-1] != '(':
                postfix += stack.pop()
            stack.pop() # Pop the opening parenthesis from the stack
        # If the character is an operator, pop operators from the stack and
add them to the postfix string until an operator with lower precedence is
encountered, then push the current operator to the stack
        elif is_operator(char):
            while stack and stack[-1] != '(' and precedence(char) <=
precedence(stack[-1]):
                postfix += stack.pop()
            stack.append(char)
    # Pop any remaining operators from the stack and add them to the postfix
string
    while stack:
```

```
        postfix += stack.pop()
    return postfix
# Test the function
expression = input("Enter an infix expression: ")
postfix = infix_to_postfix(expression)
print("Postfix expression:", postfix)
```

2. Write a Python program to evaluate postfix expression using stack.
Ans:[562+*124/-]

```
Operators =set(['*','-','+','%','/','^'])
def evaluate_postfix(expression)
     stack=[]
     for i in expression:
        if i not in Operators:
            stack.append(i)
        else:
            a=stack.pop()
            b=stack.pop()
            if i=='+':
                res=int(b)+int(a)
            elif i=='-':
                res=int(b)-int(a)
            elif i=='*':
                res=int(b)*int(a)
            elif i=='%':
                res=int(b)%int(a)
            elif i=='/':
                res=int(b)/int(a)
            elif i=='^':
                res=int(b)^int(a)

                stack.append(res)
            return(res)
        expression=input('enter postfix expression:')
        print('postfix evaluation result is:',evaluate_postfix(expression))
```

3. Write a Python program for dynamic implementation of Singly Linked List to perform following operations:
a. Create
b. Display
c. Search
Ans:

```
class node:
    def init(self,data):
        self.data=data
        self.next=none
```

```
class LinkedList:
    def init(self):
        self.head=none

    def create(self):
        ele=int(input("enter elemnt to insert:"))
        new_node=node(self)
        if self.head is none:
                 self.head=new_node
        else:
                 q=self.head
                 while(q.next):
                         q=q.next
                 q.next=new_node

def printSLL(self):
    q=self.head
    while(q):
        print(q.data,end='->')
        print("none")

def searchSLL(self,x):
 q=self.head
    while q!=none:
        if q.data==x:
            return True
        q=q.next
    return False
sll=LinkedList()
ch=0
while ch!=5:
        print("SLL Menu")
        print("1.create")
        print("2.display")
        print("3.search")
        print("4.exit")
        ch=int(input("enter your choice:"))
        if ch==1:
            n=int(input("how many numbers you wnat to enter?"))
            for i in range(0,n):
                sll.create()
        if ch==2:
            sll.printSLL()
        if ch==3:
            ele=int(input("enter element to search:"))
            if sll.searchSLL(ele):
                print("element found")
```

```
            else:
                print("element not found")
        if ch==4:
            break
```

4. Write a Python program for dynamic implementation of Doubly Circular Linked List to perform following operations:
a. Create
b. Display
ans:

```
class node:
    def __init__(self,data):
        self.data=data
        self.next=none
        self.prev=none
class DoublyCircularLinkedList:
    def __init__(self):
        self.head=none
    def create(self,data):
        new_node=node(data)
        if not self.head:
          self.head=new_node
          new_node.next=new_node.prev=self.head
        else:
            temp=self.head.prev
            temp.next=new_node
            new_node.prev=temp
            new_node.next=self.head
            self.head.prev=new_node
    def printDCLL(self):
        q=self.head
        while(q!=self.tail):
            print(q.data,end='<_>')
            q=q.next
        print(q.data,'<_>',q.next.data)
dcll=DoublyCircularLinkedList()
dcll.create(1)
dcll.create(2)
dcll.create(3)
dcll.display()
```

5. Write a Python program for static implementation of linear queue to perform following operations:
a. init
b. enqueue
c. dequeue
d. isEmpty
e. isFull

ans:

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

    def enqueue(self, value):
        if self.isFull():
            print("Queue is full.")
        else:
            if self.front == -1:
                self.front = 0
            self.rear += 1
            self.queue[self.rear] = value
            print(f"Enqueued {value} to the queue.")

    def dequeue(self):
        if self.isEmpty():
            print("Queue is empty.")
        else:
            value = self.queue[self.front]
            self.queue[self.front] = None
            self.front += 1
            if self.front > self.rear:
                self.front = -1
                self.rear = -1
            print(f"Dequeued {value} from the queue.")

    def isEmpty(self):
 if self.front == -1:
            return True
        else:
            return False

    def isFull(self):
        if self.rear == self.size - 1:
            return True
        else:
            return False


size = int(input("Enter the size of the queue: "))
queue = Queue(size)

while True:
    print("\n********** MENU **********")
    print("1. Enqueue")
```

```
    print("2. Dequeue")
    print("3. Check if queue is empty")
    print("4. Check if queue is full")
    print("5. Exit")

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

    if choice == 1:
        value = input("Enter the value to enqueue: ")
        queue.enqueue(value)

    elif choice == 2:
queue.dequeue()

    elif choice == 3:
        if queue.isEmpty():
            print("Queue is empty.")
        else:
            print("Queue is not empty.")

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

    elif choice == 5:
        break

    else:
        print("Invalid choice. Please enter a valid option from the menu.")
```

Write a Python program for dynamic implementation of Singly Linked List to perform following operations:
a. Create
b. Display
c. Merge
ans:

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

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

    def display(self):
        current = self.head
```

```
        while current:
            print(current.data, end=' ')
            current = current.next

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

def merge_lists(list1, list2):
    merged_list = SinglyLinkedList()
    current1 = list1.head
    current2 = list2.head
    while current1 and current2:
        if current1.data < current2.data:
            merged_list.append(current1.data)
            current1 = current1.next
        else:
 merged_list.append(current2.data)
            current2 = current2.next

    while current1:
        merged_list.append(current1.data)
        current1 = current1.next

    while current2:
        merged_list.append(current2.data)
        current2 = current2.next

    return merged_list

# Example Usage
# Create linked lists
list1 = SinglyLinkedList()
list1.append(1)
list1.append(3)
list1.append(5)
list2 = SinglyLinkedList()
list2.append(2)
list2.append(4)
list2.append(6)

# Display lists
```

```
print("List 1:")
list1.display()
print("\nList 2:")
list2.display()

# Merge lists
merged = merge_lists(list1, list2)
print("\nMerged List:")
merged.display()
```

Write a Python program for dynamic implementation of linear queue to perform following operations:
a. init
b. enqueue
c. dequeue
d. isEmpty
ans:

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

    def isEmpty(self):
        return self.front == -1 and self.rear == -1

    def enqueue(self, value):
        if (self.rear + 1) % self.max_size == self.front:
            print("Queue is full")
        elif self.isEmpty():
            self.front = self.rear = 0
            self.queue[self.rear] = value
        else:
            self.rear = (self.rear + 1) % self.max_size
            self.queue[self.rear] = value

    def dequeue(self):
        if self.isEmpty():
            print("Queue is empty")
        elif self.front == self.rear:
            self.front = self.rear = -1
        else:
            self.front = (self.front + 1) % self.max_size

    def display(self):
        if self.isEmpty():
            print("Queue is empty")
```

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

# Example Usage
queue = LinearQueue(5)
queue.enqueue(1)
queue.enqueue(2)
queue.enqueue(3)
queue.display()
queue.dequeue()
queue.display()
```

8. Write a Python program for static implementation of stack to perform following operations:
a. init
b. push
c. pop
d. isEmpty
e. isFull
ans:

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

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

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

    def push(self, value):
        if self.isFull():
            print("Stack is full")
        else:
            self.top += 1
            self.stack[self.top] = value

    def pop(self):
        if self.isEmpty():
            print("Stack is empty")
```

```
        else:
            self.top -= 1

    def display(self):
        if self.isEmpty():
            print("Stack is empty")
        else:
            for i in range(self.top, -1, -1):
print(self.stack[i])

# Example Usage
stack = StaticStack(5)
stack.push(1)
stack.push(2)
stack.push(3)
stack.display()
stack.pop()
stack.display()
```

9. Write a Python program for dynamic implementation of stack to perform following operations:
a. init
b. push
c. pop
d. isEmpty
ans:

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

class DynamicStack:
    def __init__(self):
        self.top = None

    def isEmpty(self):
        return self.top is None

    def push(self, data):
        new_node = Node(data)
        new_node.next = self.top
        self.top = new_node

    def pop(self):
        if self.isEmpty():
            print("Stack is empty")
        else:
            popped = self.top.data
            self.top = self.top.next
            return popped
```

```
    def display(self):
current = self.top
        while current:
            print(current.data)
            current = current.next

# Example Usage
stack = DynamicStack()
stack.push(1)
stack.push(2)
stack.push(3)
stack.display()
stack.pop()
stack.display()
```

10. Write a Python program for dynamic implementation of Singly Linked List to perform following operations:
a. Create
b. Display
c. Sort
ans:

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

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

    def create(self, data):
        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

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

    def sort(self):
        if self.head is None:
```

```
            return
        else:
            current = self.head
            while current:
                next_node = current.next
                while next_node:
                    if current.data > next_node.data:
                        current.data, next_node.data = next_node.data,
current.data
                    next_node = next_node.next
                current = current.next

# Example Usage
sll = SinglyLinkedList()
sll.create(3)
sll.create(1)
sll.create(2)
print("Singly Linked List before sorting:")
sll.display()
sll.sort()
print("Singly Linked List after sorting:")
sll.display()
```