

DSA Part B: Mid Term Journal Submission

MCA – I (Sem – I)

Academic Year 2024-2025 (Term: Aug - Dec 2024)

Lab: Data Structure and Algorithm

CO Mapped=CO4 BT Level 3=Apply and 6=Create

SR NO	Programs	Date	Sign
1	Apply the DDL algorithm and Create doubly linked list and perform the following operations: insert from head, insert from tail , insert from specified position and display	7/10/24	
2	Apply the DDL algorithm and Create doubly linked list and perform the following operations: delete from head, delete from tail , delete from specified position and display	8/10/24	
3	Apply the Circular SLL algorithm and Create Circular Singly linked list and perform the following operations: insert from head, insert from tail , insert from specified position and display	9/10/24	
4	Apply the Circular SLL algorithm and Create Circular Singly linked list and perform the following operations: delete from head, delete from tail , delete from specified position and display	17/10/24	
5	Apply the Circular DLL algorithm and Create Circular Doubly linked list and perform the following operations: insert from head, insert from tail , insert from specified position and display	18/10/24	
6	Apply the Circular DLL algorithm and Create Circular Doubly linked list and perform the following operations: delete from head, delete from tail , delete from specified position and display	21/10/24	
7	Apply Stack Algorithm and create Stack using array and perform following operations: Push, POP, Peek	22/10/24	
8	Apply Stack Algorithm and create Stack using linked list and perform following operations: Push, POP, Peek	23/10/24	

Practical 1:

Apply the DDL algorithm and Create a doubly linked list and perform the following operations: insert from head, insert from tail, insert from specified position and display

Solution:

```
class Node:
```

```
    def __init__(self,data):
```

```
        self.data=data
```

```
        self.prev=None
```

```
        self.next=None
```

```
class DLL:
```

```
    def __init__(self):
```

```
        self.head=None
```

```
        self.tail=None
```

```
    def insert_head(self,data):
```

```
        newNode=Node(data)
```

```
        if self.head is None:
```

```
            self.head=newNode
```

```
            self.tail=newNode
```

```
        else:
```

```
            newNode.next=self.head
```

```
            self.head.prev=newNode
```

```
            self.head=newNode
```

```
    def insert_tail(self,data):
```

```
        newNode=Node(data)
```

```
        if self.head is None:
```

```
            self.head=newNode
```

```
            self.tail=newNode
```

```
        else:
```

```
            self.tail.next=newNode
```

```
newNode.prev=self.tail  
self.tail=newNode
```

```
def insert_spec(self,data,loc):  
    newNode=Node(data)  
    if self.head is None:  
        self.head=newNode  
        self.tail=newNode  
    elif loc==1:  
        self.insert_head(data)  
    else:  
        cNode=self.head  
        cLoc=1  
        while cNode is not None and cLoc<loc-1:  
            cNode=cNode.next  
            cLoc+=1  
        newNode.next=cNode.next  
        cNode.next.prev=newNode  
        newNode.prev=cNode  
        cNode.next=newNode
```

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

```
dll=DLL()  
dll.insert_head(10)  
dll.insert_head(20)  
dll.display()
```

```
dll.insert_tail(40)
dll.insert_tail(50)
dll.display()
dll.insert_spec(100,2)
dll.insert_spec(75,5)
dll.display()
```

output:

```
PS E:\VS Code> python -u "e:\VS Code\DS\Mid Term\pra1.py"
20-->10-->None
20-->10-->40-->50-->None
20-->100-->10-->40-->75-->50-->None
.
```

Practical 2:

Apply the DDL algorithm and Create doubly linked list and perform the following operations: delete from head, delete from tail, delete from specified position and display

Solution:

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

    def insert_head(self,data):
        newNode=Node(data)
        if self.head is None:
            self.head=newNode
            self.tail=newNode
        else:
            newNode.next=self.head
            self.head.prev=newNode
            self.head=newNode

    def del_head(self):
        if self.head is None:
            print("Linked list is Empty")
        else:
            self.head=self.head.next
            self.head.prev=None
```

```

def del_tail(self):
    if self.head is None:
        print("Linked list is Empty")
    else:
        self.tail=self.tail.prev
        self.tail.next.prev=None
        self.tail.next=None

def del_spec(self,loc):
    if self.head is None:
        print("Linked list is Empty")
    elif loc==1:
        self.del_head()
    else:
        cNode=self.head
        cLoc=1
        while cNode is not None and cLoc<loc-1:
            cNode=cNode.next
            cLoc+=1
        if cNode.next.next:
            cNode.next=cNode.next.next
            cNode.next.prev=cNode
        else:
            cNode.next=None

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

```

```
dll=DLL()
dll.insert_head(10)
dll.insert_head(20)
dll.insert_head(30)
dll.insert_head(40)
dll.insert_head(50)
dll.display()
dll.del_head()
dll.display()
dll.del_tail()
dll.display()
dll.del_spec(3)
dll.display()
```

Output:

```
PS E:\VS Code> python -u "e:\VS Code\DS\Mid Term\pra2.py"
50-->40-->30-->20-->10-->None
40-->30-->20-->10-->None
40-->30-->20-->None
40-->30-->None
```

Practical 3:

Apply the SLL algorithm and Create Circular Singly linked list and perform the following operations: insert from head, insert from tail, insert from specified position and display

Solution:

```
class node:
```

```
    def __init__(self,data):
```

```
        self.data=data
```

```
        self.next=None
```

```
class circlList:
```

```
    def __init__(self):
```

```
        self.head=None
```

```
    def insert_beg(self,data):
```

```
        newNode = node(data)
```

```
        if self.head==None:
```

```
            self.head=newNode
```

```
            newNode.next=self.head
```

```
        else:
```

```
            cNode=self.head
```

```
            while cNode.next != self.head:
```

```
                cNode=cNode.next
```

```
            newNode.next=self.head
```

```
            self.head=newNode
```

```
            cNode.next=self.head
```

```
    def insert_end(self,data):
```

```
        newNode=node(data)
```

```
        if self.head==None:
```

```
            self.head=newNode
```

```
            newNode.next=self.head
```

```
        else:
```



```
cNode=self.head
while cNode.next != self.head:
    cNode=cNode.next
cNode.next=newNode
newNode.next=self.head
```

```
def insert_spec(self,data,loc):
    newNode=node(data)
    if self.head==None:
        self.head=newNode
        newNode.next=self.head
    elif loc==1:
        self.insert_beg(data)
    else:
        cNode=self.head
        cLoc=1
        while cNode.next!=self.head and cLoc<loc-1:
            cNode=cNode.next
            cLoc=cLoc+1
        newNode.next=cNode.next
        cNode.next=newNode
```

```
def display(self):
    if self.head==None:
        print("Linked list is empty: ")
    else:
        cNode=self.head
        while cNode.next!=self.head:
            print(cNode.data,end="-->")
            cNode=cNode.next
        print(cNode.data)
```

```
cl = circlList()
cl.insert_beg(10)
cl.insert_beg(20)
cl.display()
cl.insert_end(40)
cl.insert_end(50)
cl.display()
cl.insert_spec(100,2)
cl.insert_spec(78,4)
cl.display()
```

Output:

```
PS E:\VS Code> python -u "e:\VS Code\DS\Mid Term\prac3.py"
20-->10
20-->10-->40-->50
20-->100-->10-->40-->78-->50
PS E:\VS Code>
```

Practical 4:

Apply the Circular SLL algorithm and Create Circular Singly linked list and perform the following operations: delete from head, delete from tail , delete from specified position and display

Solution:

```
class node:
```

```
    def __init__(self, data):
```

```
        self.data = data
```

```
        self.next = None
```

```
class circlList:
```

```
    def __init__(self):
```

```
        self.head = None
```

```
    def insert_beg(self, data):
```

```
        newNode = node(data)
```

```
        if self.head is None:
```

```
            self.head = newNode
```

```
            newNode.next = self.head
```

```
        else:
```

```
            cNode = self.head
```

```
            while cNode.next != self.head:
```

```
                cNode = cNode.next
```

```
            newNode.next = self.head
```

```
            self.head = newNode
```

```
            cNode.next = self.head
```

```
    def delete_spec(self, loc):
```

```
        if self.head is None:
```

```

        print("Linked list is already empty")
        return
    if self.head.next == self.head and loc == 1:
        self.head = None
        print("After deletion, the list is empty")
        return
    if loc == 1:
        self.delete_beg()
        return
    cNode = self.head
    cLoc = 1
    while cNode.next != self.head and cLoc < loc - 1:
        cNode = cNode.next
        cLoc += 1
    if cNode.next == self.head:
        print("Position out of range")
        return
    cNode.next = cNode.next.next

def delete_beg(self):
    if self.head is None:
        print("Linked list is already empty")
        return
    if self.head.next == self.head:
        self.head = None
        print("After deletion, the list is empty")
        return
    cNode = self.head

```

```
while cNode.next != self.head:
```

```
    cNode = cNode.next
```

```
self.head = self.head.next
```

```
cNode.next = self.head
```

```
def delete_end(self):
```

```
    if self.head is None:
```

```
        print("Linked list is already empty")
```

```
        return
```

```
    if self.head.next == self.head:
```

```
        self.head = None
```

```
        print("After deletion, the list is empty")
```

```
        return
```

```
    cNode = self.head
```

```
    while cNode.next.next != self.head:
```

```
        cNode = cNode.next
```

```
    cNode.next = self.head
```

```
def display(self):
```

```
    if self.head is None:
```

```
        print("Linked list is empty")
```

```
        return
```

```
    cNode = self.head
```

```
    while cNode.next != self.head:
```

```
        print(cNode.data, end=" --> ")
```

```
        cNode = cNode.next
```

```
    print(cNode.data)
```

```
cl = circlList()
cl.insert_beg(10)
cl.insert_beg(20)
cl.insert_beg(30)
cl.insert_beg(40)
cl.insert_beg(50)
cl.display()
cl.delete_beg()
cl.display()
cl.delete_end()
cl.display()
cl.delete_spec(3)
cl.display()
```

Output:

```
PS E:\VS Code> python -u "e:\VS Code\DS\Mid Term\prac4.py"
50 --> 40 --> 30 --> 20 --> 10
40 --> 30 --> 20 --> 10
40 --> 30 --> 20
40 --> 30
```

Practical 5:

Apply the circular DLL algorithm and Create Circular Doubly linked list and perform the following operations: insert from head, insert from tail, insert from specified position and display

Solution:

Apply the Circular DLL algorithm and Create Circular Doubly
linked list and perform the following operations: insert from
head, insert from tail , insert from specified position and display

```
class node:
```

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

```
class circlList:
```

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

```
    def insert_beg(self,data):  
        newNode = node(data)  
        if self.head==None:  
            self.head=newNode  
            newNode.next=self.head  
            newNode.prev=self.head  
        else:  
            cNode=self.head  
            while cNode.next != self.head:
```

```
    cNode=cNode.next
newNode.next=self.head
newNode.prev=cNode
self.head=newNode
cNode.next=self.head
```

```
def insert_end(self,data):
    newNode=node(data)
    if self.head==None:
        self.head=newNode
        newNode.next=self.head
    else:
        cNode=self.head
        while cNode.next != self.head:
            cNode=cNode.next
        newNode.prev=cNode
        cNode.next=newNode
        newNode.next=self.head
```

```
def insert_spec(self,data,loc):
    newNode=node(data)
    if self.head==None:
        self.head=newNode
        newNode.next=self.head
        newNode.prev=self.head
    elif loc==1:
        self.insert_beg(data)
    else:
```



```
cNode=self.head
cLoc=1
while cNode.next!=self.head and cLoc<loc-1:
    cNode=cNode.next
    cLoc=cLoc+1
newNode.next=cNode.next
cNode.next.prev=newNode
cNode.next=newNode
newNode.prev=cNode
```

```
def display(self):
    if self.head==None:
        print("Linked list is empty: ")
    else:
        cNode=self.head
        while cNode.next!=self.head:
            print(cNode.data,end="<-->")
            cNode=cNode.next
        print(cNode.data)
```

```
def display_rev(self):
    if self.head == None:
        print("Linked list is empty")
    else:
        cNode = self.head
        while cNode.next != self.head:
            cNode = cNode.next
        temp = cNode
```

```

while temp != self.head:
    print(temp.data, end="<-->")
    temp = temp.prev
print(temp.data)

```

```

cl = circlList()
cl.insert_beg(10)
cl.insert_beg(20)
cl.display()
cl.insert_end(40)
cl.insert_end(50)
cl.display()
cl.insert_spec(100,2)
cl.insert_spec(78,4)
cl.display()
cl.display_rev()

```

Output:

```

PS E:\VS Code> python -u "e:\VS Code\DS\Mid Term\prac5.py"
20<-->10
20<-->10<-->40<-->50
20<-->100<-->10<-->78<-->40<-->50
50<-->40<-->78<-->10<-->100<-->20

```

Practical 6:

Apply the Circular DLL algorithm and Create Circular Doubly linked list and perform the following operations: delete from head, delete from tail, delete from specified position and display

Solution:

```
class node:
```

```
    def __init__(self, data):
```

```
        self.data = data
```

```
        self.next = None
```

```
        self.prev = None
```

```
class circlList:
```

```
    def __init__(self):
```

```
        self.head = None
```

```
    def insert_beg(self, data):
```

```
        newNode = node(data)
```

```
        if self.head == None:
```

```
            self.head = newNode
```

```
            newNode.next = self.head
```

```
            newNode.prev = self.head
```

```
        else:
```

```
            cNode = self.head
```

```
            while cNode.next != self.head:
```

```
                cNode = cNode.next
```

```
            newNode.next = self.head
```

```
            newNode.prev = cNode
```

```
            self.head.prev = newNode
```

```
            self.head = newNode
```

```
cNode.next = self.head
```

```
def delete_head(self):  
    if self.head == None:  
        print("Linked list is already empty")  
    elif self.head.next == self.head:  
        self.head = None  
    else:  
        cNode = self.head  
        while cNode.next != self.head:  
            cNode = cNode.next  
        self.head = self.head.next  
        self.head.prev = cNode  
        cNode.next = self.head
```

```
def delete_tail(self):  
    if self.head == None:  
        print("Linked list is already empty")  
    elif self.head.next == self.head:  
        self.head = None  
    else:  
        cNode = self.head  
        while cNode.next != self.head:  
            cNode = cNode.next  
        cNode.prev.next = self.head  
        self.head.prev = cNode.prev
```

```
def delete_spec(self, loc):
```

```

if self.head == None:
    print("Linked list is already empty")
elif loc == 1:
    self.delete_head()
else:
    cNode = self.head
    cLoc = 1
    while cNode.next != self.head and cLoc < loc:
        cNode = cNode.next
        cLoc += 1
    if cNode == self.head:
        print("Location exceeds the length of the list")
    else:
        cNode.prev.next = cNode.next
        cNode.next.prev = cNode.prev

```

```

def display(self):
    if self.head == None:
        print("Linked list is empty")
    else:
        cNode = self.head
        while cNode.next != self.head:
            print(cNode.data, end="<-->")
            cNode = cNode.next
        print(cNode.data)

```

```

cl = circlList()

```

```
cl.insert_beg(10)
cl.insert_beg(20)
cl.insert_beg(30)
cl.insert_beg(40)
cl.insert_beg(50)
cl.display()
cl.delete_head()
cl.display()
cl.delete_tail()
cl.display()
cl.delete_spec(2)
cl.display()
```

Output:

```
PS E:\VS Code> python -u "e:\VS Code\DS\Mid Term\prac6.py"
50<-->40<-->30<-->20<-->10
40<-->30<-->20<-->10
40<-->30<-->20
40<-->20
```

Practical 7:

Apply Stack Algorithm and create Stack using array and perform following operations: Push, POP, Peek

Solution:

```
class Stack:
```

```
    def __init__(self):
```

```
        self.stack = []
```

```
    def push(self, item):
```

```
        self.stack.append(item)
```

```
    def isEmpty(self):
```

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

```
    def pop(self):
```

```
        if self.isEmpty():
```

```
            return("Underflow - Stack is empty")
```

```
        else:
```

```
            return self.stack.pop()
```

```
    def peek(self):
```

```
        if self.isEmpty():
```

```
            return("Underflow - Stack is empty")
```

```
        else:
```

```
            return self.stack[-1]
```

```
    def display(self):
```

```
        if self.isEmpty():
```

```
        return("Underflow - Stack is empty")
    else:
        print(self.stack)

s=Stack()
s.push(10)
s.push(20)
s.push(30)
s.display()
print("Stack Top : ",s.peak())
print("Popped :",s.pop())
print("Popped :",s.pop())
s.display()
```

Output:

```
PS E:\VS Code> python -u "e:\VS Code\DS\Mid Term\prac7.py"
[10, 20, 30]
Stack Top : 30
Popped : 30
Popped : 20
[10]
```


Practical 8:

Apply Stack Algorithm and create Stack using linked list and perform following operations: Push, POP, Peek

Solution:

Apply Stack Algorithm and create Stack using linked list and

perform following operations: Push, POP, Peek

```
class Node:
```

```
    def __init__(self, data):
```

```
        self.data = data
```

```
        self.next=None
```

```
class Stack:
```

```
    def __init__(self):
```

```
        self.top=None
```

```
    def push(self,data):
```

```
        newNode=Node(data)
```

```
        if self.top is None:
```

```
            self.top=newNode
```

```
        else:
```

```
            newNode.next=self.top
```

```
            self.top=newNode
```

```
    def pop(self):
```

```
        if self.top is None:
```

```
            return "Stack is empty"
```

```
        else:
```

```
            popped_node=self.top
```

```
self.top=self.top.next
popped_node.next=None
return popped_node.data
```

```
def peek(self):
    if self.top is None:
        return "Stack is empty"
    else:
        return self.top.data
```

```
def display(self):
    if self.top is None:
        print("Stack is Empty")
    else:
        temp = self.top
        print("----")
        while temp:
            print(temp.data)
            print("----")
            temp=temp.next
```

```
s=Stack()
s.push(10)
s.push(20)
s.push(30)
s.push(40)
print("After pushing: ")
s.display()
```

```
print("Popped Element :",s.pop())  
print("After popping: ")  
s.display()  
print("Stack Top :",s.peek())
```

Output:

```
PS E:\VS Code> python -u "e:\VS Code\DS\Mid Term\prac8.py"  
After pushing:  
-----  
40  
-----  
30  
-----  
20  
-----  
10  
-----  
Popped Element : 40  
After popping:  
-----  
30  
-----  
20  
-----  
10  
-----  
Stack Top : 30
```