7.1 Implementation of Doubly Linked list

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

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

    #To Append a Node
    def append(self,data):
        newnode=Node(data)
        if self.head==None:
            self.head=newnode
            self.head.prev=None
            self.head.next=None
        else:
            last = self.head
            while (last.next is not None):
                last = last.next

            last.next = newnode
            newnode.prev = last

    #To Display the Nodes
    def display(self):
        current=self.head
        if self.head==None:
            print("List is Empty")
            return
        print("Node of Doubly Linked List")
        while current != None:
            print(current.data)
            current=current.next

    #To Delete a Node
    def deletenode(self,key):
        temp=self.head
        if temp is not None:
            if temp.data==key:
                self.head=temp.next
                temp=None
                return
        while temp is not None:
            if temp.data is key:
                break
            prev=temp
            temp=temp.next
        if temp==None:
            return
        prev.next=temp.next
        temp=None

d=doubly()
d.append(1)
d.append(2)
d.append(3)
d.append(4)
d.append(5)
d.append(6)
d.display()
d.deletenode(2)
print("After deleting elements ")
d.display()
```

Output:

```
Node of Doubly Linked List
1
2
3
4
5
6
After deleting elements
Node of Doubly Linked List
1
3
4
5
6
```

7.2 Implementation of Circular linked list

```
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
class Circullarll:
    def __init__(self):
        self.head=Node(None)
        self.head.next=self.head
    def append(self,data):
        newnode=Node(data)
        if (self.head == None):
            self.head = newnode
            newnode.next = self.head
            return
        else:
            temp = self.head
            while (temp.next != self.head):
                temp = temp.next
            temp.next = newnode
            newnode.next = self.head

    def display(self):
        current=self.head
        if self.head is None:
            print("List is Empty")
            return
        else:
            print("Node in Linkedlist are")
            # print(current.data)
            while (current.next!=self.head):
                current=current.next
                print (current.data)

    def deletenode(self, key):
        temp = self.head
        if temp.next is not temp:
            if temp.data == key:
                self.head = temp.next
                temp = None
                return
        while temp.next is not self.head:
            if temp.data is key:
                break
            prev = temp
            temp = temp.next
        if temp == self.head:
            return
        prev.next = temp.next
        # temp = None


a=Circullarll()
a.append(1)
a.append(2)
a.append(3)
a.append(4)
a.display()
a.deletenode(2)
print("After deleting the element")
a.display()
```

Output:

```
Node in Linkedlist are
1
2
3
4
After deleting the element
Node in Linkedlist are
1
3
4
```

8.1 Implement Stack Data Structure

```
class Stack:
  def __init__(self):
    self.items=[]
  def isEmpty(self):
    return self.items==[]
  def push(self):
    data = int(input("Enter the number : "))
    self.items.append(data)
```

```
  def pop(self):
      return self.items.pop()
  def display(self):
      if self.items==[]:
          print("List is empty")
      else:
          for i in self.items:
              print(i)
s=Stack()
while True:
  print("1. Push ")
  print("2. Pop")
  print("3. Display")
  print("4. Quit")
  ch=input("Enter the value for operation: ")
  if ch=="1":
      s.push()
  elif ch=="2":
      if s.isEmpty():
          print("Stack is empty")
          break
      else:
          print("Removed value: ",s.pop())
  elif ch=="3":
       s.display()
  elif ch=="4":
      break
```

Output:

```
1. Push
2. Pop
3. Display
4. Quit
Enter the value for operation: 1
Enter the number : 3
1. Push
2. Pop
3. Display
4. Quit
Enter the value for operation: 1
Enter the number : 5
1. Push
2. Pop
3. Display
4. Quit
Enter the value for operation: 3
3
5
1. Push
2. Pop
3. Display
4. Quit
Enter the value for operation: 2
Removed value:  5
1. Push
```

```
2. Pop
3. Display
4. Quit
Enter the value for operation: 4
```

8.2 Implement bracket matching using stack.

```
open_list=["[", "{", "("]
close_list=["]", "}", ")"]
def check(mystr):
   stack=[]
   for i in mystr:
      if i in open_list:
         stack.append(i)
      elif i in close_list:
         por=close_list.index(i)
         if ((len(stack)>=0) and (open_list[por]==stack[len(stack)-1])):
            stack.pop()
         else:
            return "Unbalanced"
   if len(stack)==0:
      return "Balanced"
   else:
      return "Unbalanced"

string = input("Enter the brackets : ")
print(string, " is: ",check(string))
```

Output1:

```
Enter the brackets : ({})[]{}
({})[]{}  is:  Balanced
```

Output2:

```
Enter the brackets : ({)}[]{}
({)}[]{}  is:  Unbalanced
```

9.1 Program to demonstrate recursive operations (Factorial/ Fibonacci).

a)Fibonnaci

```
def fib(n):
   if n<=1:
      return n
   else:
      return fib(n-1)+fib(n-2)

n = int(input("Enter the number: "))
if n<0:
   print("Enter the Positive number..")
else:
    fib_ser = []
    for i in range(n):
       fib_ser.append(fib(i))
    print("The fibonnaci series is :", fib_ser )
    print("The fibonnaci of given number is :", fib_ser[-1])
```

Output:

```
 Enter the number: 5
Enter the number: 5
The fibonnaci series is : [0, 1, 1, 2, 3]
The fibonnaci of given number is : 3
```

b) Factorial

```
def factorial(x):
    if x == 1:
        return 1
    else:
        return (x * factorial(x-1))
num = int(input("Enter a number: "))
result = factorial(num)
print("The factorial of", num, "is", result)
```

Output:

```
Enter a number: 5
The factorial of 5 is 120
```

9.2 Implement solution for Towers of Hanoi.

```
 def TOH (n, spole, dpole, ipole):
    if (n == 1):
        print("move disc 1 from pole", spole, "to pole", dpole)
        return
    TOH (n-1, spole, ipole, dpole)
    print("move disc", n, "from pole", spole, "to pole", dpole)
    TOH (n-1, ipole, dpole, spole)
n = 3
TOH(n, 'A', 'B', 'C')
```

Output:

```
move disc 1 from pole A to pole B
move disc 2 from pole A to pole C
move disc 1 from pole B to pole C
move disc 3 from pole A to pole B
move disc 1 from pole C to pole A
move disc 2 from pole C to pole B
move disc 1 from pole A to pole B
```