LAB MANUAL

II Year B. Tech II- Semester

MECHANICAL ENGINEERING

AY: 2022-23

DATA STRUCTURES USING PYTHON LAB

R20A0384

Prepared by:
Mr. CH. NARAYANA MURTHY
Assistant Professor

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

DEPARTMENT OF MECHANICAL ENGINEERING

(Autonomous Institution-UGC, Govt. of India)
Secunderabad-500100, Telangana State, India.
www.mrcet.ac.in

DEPARTMENT OF MECHANICAL ENGINEERING

Vision

To acknowledge quality education and instill high patterns of discipline making the students technologically superior and ethically strong which involves the improvement in the quality of life in human race.

Mission

- To achieve and impart holistic technical education using the best of infrastructure, outstanding technical and teaching expertise to establish the students in to competent and confident engineers.
- Evolving the center of excellence through creative and innovative teaching learning practices for promoting academic achievement to produce internationally accepted competitive and world class professionals.

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

ANALYTICAL SKILLS

To facilitate the graduates with the ability to visualize, gather information, articulate, analyze, solve complex problems, and make decisions. These are essential to address the challenges of complex and computation intensive problems increasing their productivity.

TECHNICAL SKILLS

To facilitate the graduates with the technical skills that prepare them for immediate employment and pursue certification providing a deeper understanding of the technology in advanced areas of computer science and related fields, thus encouraging to pursue higher education and research based on their interest.

SOFT SKILLS

To facilitate the graduates with the soft skills that include fulfilling the mission, setting goals, showing self-confidence by communicating effectively, having a positive attitude, get involved in team-work, being a leader, managing their career and their life.

PROFESSIONAL ETHICS

To facilitate the graduates with the knowledge of professional and ethical responsibilities by paying attention to grooming, being conservative with style, following dress codes, safety codes, and adapting themselves to technological advancements.

PROGRAM SPECIFIC OUTCOMES (PSOs)

After the completion of the course, B. Tech Computer Science and Engineering, the graduates will have the following Program Specific Outcomes:

1. Fundamentals and critical knowledge of the Computer System:- Able to Understand the working principles of the computer System and its components , Apply the knowledge to build, asses, and analyze the software and hardware aspects of it .

2. The comprehensive and Applicative knowledge of Software Development: Comprehensive skills of Programming Languages, Software process models, methodologies, and able to plan, develop, test, analyze, and manage the software and hardware intensive systems in heterogeneous platforms individually or working in teams.

3. Applications of Computing Domain & Research: Able to use the professional, managerial, interdisciplinary skill set, and domain specific tools in development processes, identify the research gaps, and provide innovative solutions to them.

PROGRAM OUTCOMES (POs)

Engineering Graduates should possess the following:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. **Design / development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multi disciplinary environments.

12. **Life- long learning:** Recognize the need and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY
Maisammaguda, Dhulapally Post, Via Hakimpet, Secunderabad – 500100

GENERAL LABORATORY INSTRUCTIONS

1.Students are advised to come to the laboratory at least 5 minutes before (to the starting time), those who come after 5 minutes will not be allowed into the lab.

2.Plan your task properly much before to the commencement, come prepared to the lab with the synopsis / program / experiment details.

3.Student should enter into the laboratory with:

a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.

b. Laboratory Record updated up to the last session experiments and other utensils (if any) needed in the lab.

c. Proper Dress code and Identity card.

4.Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty.

5.Execute your task in the laboratory, and record the results / output in the lab observation note book, and get certified by the concerned faculty.

6.All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.

7.Computer labs are established with sophisticated and high end branded systems, which should be utilized properly.

8.Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviors with the staff and systems etc., will attract severe punishment.

9.Students must take the permission of the faculty in case of any urgency to go out; if anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.

10.Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

HEAD OF THE DEPARTMENT **PRINCIPAL**

INDEX

1.Write a Python program for class, Flower, that has three instance variables of type str, int, and float, that respectively represent the name of the flower, its number of petals, and its price. Your class must include a constructor method that initializes each variable to an appropriate value, and your class should include methods for setting the value of each type, and retrieving the value of each type.

Program:-

```
class Flower:
    #Common base class for all Flowers
        def __init__(self, petalName, petalNumber, petalPrice): self.name =
        petalName
        self.petals = petalNumber self.price = petalPrice

        def setName(self, petalName):
            self.name = petalName
        def setPetals(self, petalNumber):
             self.petals = petalNumber

        def setPrice(self, petalPrice):
            self.price = petalPrice

        def getName(self):
            return self.name
        def getPetals(self):
            return self.petals
        def getPrice(self):
            return self.price


#This would create first object of Flower
class f1 = Flower("Sunflower", 2, 1000)
print ("Flower Details:")
print ("Name: ", f1.getName())
print ("Number of petals:", f1.getPetals())
print ("Price:",f1.getPrice())
print ("\n")

#This would create second object of Flower
class f2 = Flower("Rose", 5, 2000)
f2.setPrice(3333)
f2.setPetals(6)
print ("Flower Details:")
print ("Name: ", f2.getName())
print ("Number of petals:", f2.getPetals())
print ("Price:",f2.getPrice())
```

Output:

Signature of the Faculty

2.Develop an inheritance hierarchy based upon a Polygon class that has abstract methods area( ) and perimeter( ). Implement classes Triangle, Quadrilateral, Pentagon, that extend this base class, with the obvious meanings for the area( ) and perimeter( ) methods. Write a simple program that allows users to create polygons of the various types and input their geometric dimensions, and the program then outputs their area and perimeter.

Program:

```
from abc import abstractmethod, ABCMeta import math
class Polygon(metaclass = ABCMeta):
def __init__(self, side_lengths = [1,1,1], num_sides = 3):
        self._side_lengths = side_lengths
        self._num_sizes = 3

@abstractmethod
def area(self):
        pass

@abstractmethod
def perimeter(self):
        pass

def __repr__(self):
return (str(self._side_lengths))


class Triangle(Polygon):
        def __init__(self, side_lengths):
                super().__init__(side_lengths, 3)
                self._perimeter = self.perimeter()
                self._area = self.area()

def perimeter(self):
        return(sum(self._side_lengths))
def area(self):
        #Area of Triangle
        s = self._perimeter/2 product = s
        for i in self._side_lengths: product*=(s-i)
                return product**0.5

class Quadrilateral(Polygon):
def __init__(self, side_lengths):
        super().__init__(side_lengths, 4)
        self._perimeter = self.perimeter()

self._area = self.area()

def perimeter(self):
```

```
        return(sum(self._side_lengths))

def area(self):
# Area of an irregular Quadrilateral
        semiperimeter = sum(self._side_lengths) / 2
        return math.sqrt((semiperimeter - self._side_lengths[0]) * (semiperimeter -
        self._side_lengths[1]) * (semiperimeter - self._side_lengths[2]) * (semiperimeter -
self._side_lengths[3]))

class Pentagon(Polygon):
        def __init__(self, side_lengths):
        super().__init__(side_lengths, 5)
        self._perimeter = self.perimeter()
        self._area = self.area()

def perimeter(self):
        return((self._side_lengths) * 5)

def area(self):
        # Area of a regular Pentagon
  a = self._side_lengths
  return (math.sqrt(5*(5 + 2 * (math.sqrt(5)))) * a * a) / 4

#object of Triangle
t1 = Triangle([1,2,2])
print(t1.perimeter(), t1.area())

#object of Quadrilateral
q1 = Quadrilateral([1,1,1,1])
print(q1.perimeter(), q1.area())

#object of Pentagon p1 = Pentagon(1)
print(p1.perimeter(), p1.area())
```

Output:

Signature of the Faculty

3.Write a python program to implement method overloading and method overriding.

Method Overloading

Method overloading is an OOPS concept which provides ability to have several methods having the same name with in the class where the methods differ in types or number of arguments passed.

Method overloading in Python

Method overloading in its traditional sense (as defined above) as exists in other languages like method overloading in Java doesn't exist in Python.

In Python if you try to overload a function by having two or more functions having the same name but different number of arguments only the last defined function is recognized, calling any other overloaded function results in an error.

Achieving method overloading

Since using the same method name again to overload the method is not possible in Python, so achieving method overloading in Python is done by having a single method with several parameters. Then you need to check the actual number of arguments passed to the method and perform the operation accordingly.

Program:

```
class OverloadDemo:
# sum method with default as None for parameters
def sum(self, a=None, b=None, c=None):
        # When three params are passed
        if a!=None and b!=None and c!=None:
                s = a + b + c
                print('Sum = ', s)
        # When two params are passed
        elif a!=None and b!=None:
                s = a + b
                print('Sum = ', s)

od =    OverloadDemo()
od.sum(7, 8)
od.sum(7, 8, 9)
```

Output:

Signature of the Faculty

Method overriding - Polymorphism through inheritance

Method overriding provides ability to change the implementation of a method in a child class which is already defined in one of its super class. If there is a method in a super class and method having the same name and same number of arguments in a child class then the child class method is said to be overriding the parent class method.

When the method is called with parent class object, method of the parent class is executed. When method is called with child class object, method of the child class is executed. So the appropriate overridden method is called based on the object type, which is an example of Polymorphism.

Program:

```
class Person:
def     init     (self, name, age):
        self.name = name
        self.age = age

def displayData(self):
        print('In parent class displayData method')
        print(self.name)
        print(self.age)

class Employee(Person):
        def __init__(self, name, age, id):
        # calling constructor of super class
        super().__init__(name, age)
        self.empId = id

def displayData(self):
        print('In child class displayData method')
        print(self.name)
        print(self.age)
        print(self.empId)

#Person class object
person = Person('Karthik Shaurya', 26)
person.displayData()
#Employee class object
emp = Employee(''Karthik Shaurya', 26, 'E317')
emp.displayData()
```

Output:

Signature of the Faculty

4.Write a Python program to illustrate the following comprehensions:
a)List Comprehensions b) Dictionary Comprehensions
c) Set Comprehensions d) Generator Comprehensions

Comprehensions in Python
Comprehensions in Python provide us with a short and concise way to construct new sequences (such as lists, set, dictionary etc.) using sequences which have been already defined. Python supports the following 4 types of comprehensions:

a)List Comprehensions
b)Dictionary Comprehensions
c)Set Comprehensions
d)Generator Comprehensions

a)List Comprehensions:
List Comprehensions provide an elegant way to create new lists. The following is the basic structure of a list comprehension:

output_list = [output_exp for var in input_list if (var satisfies this condition)]

Note that list comprehension may or may not contain an if condition. List comprehensions can contain multiple for (nested list comprehensions).

Example: Suppose we want to create an output list which contains only the even numbers which are present in the input list. Let's see how to do this using for loop and list comprehension and decide which method suits better.

Using Loop:

```
#Constructing output list WITHOUT using List comprehensions
input_list = [1, 2, 3, 4, 4, 5, 6, 7, 7]
output_list = []
#Using loop for constructing output list
for var in input_list:
        if var % 2 == 0:
                output_list.append(var)
print("Output List using for loop:", output_list)
```

Output:

Using List Comprehension:

```
# Using List comprehensions
# for constructing output list
input_list = [1, 2, 3, 4, 4, 5, 6, 7, 7]
list_using_comp = [var for var in input_list if var % 2 == 0]
print("Output List using list comprehensions:",list_using_comp)
```

Output:

b)Dictionary Comprehensions

Extending the idea of list comprehensions, we can also create a dictionary using dictionary comprehensions. The basic structure of a dictionary comprehension looks like below.

output_dict = {key:value for (key, value) in iterable if (key, value satisfy this condition)}

Example 1: Suppose we want to create an output dictionary which contains only the odd numbers that are present in the input list as keys and their cubes as values. Let's see how to do this using for loops and dictionary comprehension.

Using Loop:

```
input_list = [1, 2, 3, 4, 5, 6, 7]
output_dict = {}
# Using loop for constructing output dictionary
for var in input_list:
    if var % 2 != 0:
        output_dict[var] = var**3
print("Output Dictionary using for loop:",output_dict)
```

Output:

Using Dictionary Comprehension:

```
# Using Dictionary comprehensions
# for constructing output dictionary
input_list = [1,2,3,4,5,6,7]
dict_using_comp = {var:var ** 3 for var in input_list if var % 2 != 0}
print("Output Dictionary using dictionary comprehensions:", dict_using_comp)
```

output:

Example 2: Given two lists containing the names of states and their corresponding capitals, construct a dictionary which maps the states with their respective capitals. Let's see how to do this using for loops and dictionary comprehension.
Using Loop:

```
state = ['Gujarat', 'Maharashtra', 'Rajasthan'] capital = ['Gandhinagar', 'Mumbai', 'Jaipur']
output_dict = {}
# Using loop for constructing output dictionary
for (key, value) in zip(state, capital):
	output_dict[key] = value
print("Output Dictionary using for loop:", output_dict)
```

Output:

Using Dictionary Comprehension

```
# Using Dictionary comprehensions
# for constructing output dictionary
state = ['Gujarat', 'Maharashtra', 'Rajasthan']
capital = ['Gandhinagar', 'Mumbai', 'Jaipur']
dict_using_comp = {key:value for (key, value) in zip(state, capital)}
print("Output Dictionary using dictionary comprehensions:",dict_using_comp)
```

c) Set Comprehensions:

Set comprehensions are pretty similar to list comprehensions. The only difference between them is that set comprehensions use curly brackets { }. Let's look at the following example to understand set comprehensions.
Example : Suppose we want to create an output set which contains only the even numbers that are present in the input list. Note that set will discard all the duplicate values. Let's see how we can do this using for loops and set comprehension.
Using Loop:

```
input_list = [1, 2, 3, 4, 4, 5, 6, 6, 6, 7, 7]
output_set = set()
# Using loop for constructing output set
for var in input_list:
	if var % 2 == 0:
		output_set.add(var)
print("Output Set using for loop:", output_set)
```

output:

Using Set Comprehension:

```
# Using Set comprehensions
# for constructing output set
input_list = [1, 2, 3, 4, 4, 5, 6, 6, 6, 7, 7]
set_using_comp = {var for var in input_list if var % 2 == 0}
print("Output Set using set comprehensions:",set_using_comp)
```

Output:

d)Generator Comprehensions:
Generator Comprehensions are very similar to list comprehensions. One difference between them is that generator comprehensions use circular brackets whereas list comprehensions use square brackets. The major difference between them is that generators don’t allocate memory for the whole list. Instead, they generate each value one by one which is why theyare memory efficient. Let’s look at the following example to understand generator comprehension:

```
input_list = [1, 2, 3, 4, 4, 5, 6, 7, 7]
output_gen = (var for var in input_list if var % 2 == 0)
print("Output values using generator comprehensions:", end = ' ')
for var in output_gen:
	print(var, end = ' ')
```

5.Write a Python program to generate the combinations of n distinct objects taken from the elements of a given list. Example: Original list: [1, 2, 3, 4, 5, 6, 7, 8, 9] Combinations of 2 distinct objects: [1, 2] [1, 3] [1, 4] [1, 5] [7, 8] [7, 9] [8, 9].

Program.

```
def combination(n, n_list):
if n<=0:
        yield [ ] return
for i in range(len(n_list)):
        c_num = n_list[i:i+1]
for a_num in combination(n-1, n_list[i+1:]):
        yield c_num + a_num

n_list = [1,2,3,4,5,6,7,8,9]
print("Original list:")
print(n_list)
n = 2
result = combination(n, n_list)
print("\nCombinations of",n,"distinct objects:")
for e in result:
        print(e)
```

Output:

6.Write a program for Linear Search and Binary search

Linear Search Program:

```
from array import *
def linear_Search(list1, n, key):
       # Searching list1 sequentially
         for i in range(0, n):
           if (list1[i] == key):
             return i
         return -1

ArraySize = int(input('Enter How many Elements to read:'))
list1 = array('i', [])
for i in range(ArraySize):
  print("Enter ",str(i+1),"Element")
  list1.append(int(input()))

key = int(input('Enter Key:'))
# Function call
n=int(len(list1)-1)
res = linear_Search(list1, n, key)
if res != -1:
  print("Element is present at index", str(res))
else:
  print("Element is not present in list")
```

Output:

Binary Search Program

```
from array import *
def binary_search(ls, n):
   low = int(0)
   high = int(len(ls)-1)
   mid = int(0)
   while low <= high:
      mid = int((high + low)/2)
      if ls[mid] < n:
         low = mid + 1
      elif ls[mid] > n:
         high = mid - 1
      else:
         return mid
   return -1
# Initial list1


ArraySize = int(input('Enter How many Elements to read:'))

list1 = array('i', [])

for i in range(ArraySize):
   print("Enter ",str(i+1),"Element")
   list1.append(int(input()))

key = int(input('Enter Key:'))
# Function call
res = binary_search(list1, key)
if res != -1:
   print("Element is present at index", str(res))
else:
   print("Element is not present in list")
```

7.Write a program to implement Bubble Sort and Selection Sort

Bubble Sort Program:

```
from array import *

def bubble_sort(list1, n):
  for j in range(len(list1) - 1):
    for i in range(len(list1) - 1):
      if list1[i] > list1[i + 1]:
        t = list1[i]
        list1[i] = list1[i+1]
        list1[i + 1] = t
  return list1

ArraySize = int(input('Enter How many Elements to read:'))

list1 = array('i', [])
for i in range(ArraySize):
  list1.append(int(input()))
print('Sorted list is ')
res=bubble_sort(list1, ArraySize)
for i in range(ArraySize):
  print(res[i])
```

Selection Sort Program:

```
from array import *

def selection_sort(lst1, n):
    for i in range(n - 1):
        mini = i
        for j in range(i+1, n):
            if lst1[mini] > lst1[j]:
                mini = j

        t = lst1[i]
        lst1[i] = lst1[mini]
        lst1[mini] = t

    return lst1


ArraySize = int(input('Enter How many Elements to read:'))

list1 = array('i', [])
for k in range(ArraySize):
    list1.append(int(input()))
print('Sorted list is ')
res = selection_sort(list1, ArraySize)
for k in range(ArraySize):
    print(res[k])
```

8.Write a program to implement Merge sort and Quick sort.

Merge Sort Program:

```
from array import *

def mergesort(a, low, high):
    temp = array('i', [])
    if low < high:
        mid = int((low + high) / 2)
        mergesort(a, low, mid)
        mergesort(a, mid + 1, high)
        #CODE FOR MERGING SUB ARRAY'S
        i = low
        j = mid + 1

        while i <= mid and j <= high:
            if a[i] <= a[j]:
                temp.append(a[i])
                i = i + 1
            else:
                temp.append(a[j])
                j = j + 1

        if i > mid:
            while j <= high:
                temp.append(a[j])
                j = j + 1
        else:
            while i <= mid:
                temp.append(a[i])
                i = i + 1
   #copying back values from temp array to main array
        k = low
        for z in temp:
            a[k] = z
            k = k+1

list1 = array('i', [])
num = int(input('Enter How many Elements to read:'))
print("Enter ",num," elements")
for y in range(num):
    list1.append(int(input()))
print('Sorted list is ')
mergesort(list1, 0, num-1)
for y in range(num):
    print(list1[y],end=" ")
```

Quick Sort Program:

```
from array import *

def partition(x, low, high):
    if low < high:

        down = low
        up = high
        pivot = down
        while down < up:
            while x[down] <= x[pivot] and down < high:
                down = down + 1
            while x[up] > x[pivot]:
                up = up-1
            if down < up:
                t = x[down]
                x[down] = x[up]
                x[up] = t

        t = x[pivot]
        x[pivot] = x[up]
        x[up] = t
    return up

def quicksort(x, low, high):
    if low < high:
        p = int(partition(x, low, high))
        quicksort(x, low, p - 1)
        quicksort(x, p + 1, high)

    return x

list1 = array('i', [])
num = int(input('Enter How many Elements to read:'))

for i in range(num):
    list1.append(int(input()))
print('Sorted list is ')
res = quicksort(list1, 0, num-1)
for i in range(num):
    print(res[i], end=" ")
```

9.Write a program to implement Stacks and Queues

Stack Program using list:

```
class Stack:
  def __init__(self):
    self.stk = []
    self.top = int(-1)
    self.max=int(3)

  def Push(self):
    if self.top == self.max-1:
      print("Stack Full")
    else:
      val = input("Enter Value to be Pushed")
      self.top = self.top+1
      self.stk.append(val)
      print(val,"Pushed on to the stack")
  def Pop(self):
    if self.top == -1:
      print("Stack is Empty")
    else:
      val = self.stk.pop(self.top)
      self.top = self.top-1
      print(val,"Poped from the stack")
  def Peek(self):
    if self.top == -1:
      print("Stack is Empty")
    else:
      print("Topest Element:",self.stk[self.top])
  def Display(self):
    if self.top == -1:
      print("Stack is Empty")
    else:
      print("Elements in the Stack are:")
      new_lst = self.stk[::-1]
      for x in new_lst:
        print("|", x, "|")
StackObj = Stack()
while True:
  print("\n****Operations On Stack ***")
  print("1.Push")
  print("2.PoP")
  print("3.Peek")
  print("4.Display")
  print("5.Exit")
  choice = int(input("Enter Your Choice:"))
  if choice == 1:
    StackObj.Push()
  elif choice == 2:
```

```
        StackObj.Pop()
    elif choice == 3:
        StackObj.Peek()
    elif choice == 4:
        StackObj.Display()
    elif choice == 5:
        exit(0)
    else:
        print("Invalid Choice! Try Again:")
```

Output:

Stack Program using linked list:

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

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

   def Push(self):
      data_in = input("Enter Value to be Pushed")
      NewNode = Node(data_in)
      NewNode.link = self.top
      self.top = NewNode
      print(data_in," Pushed on to stack")

   def Pop(self):
      temp = self.top
      if temp is not None:
            self.top = temp.link
            print(temp.data,"is Deleted from Stack")
            temp = None
      else:
         print("Stack is Empty")

   def Peek(self):
      temp = self.top
      if temp is None:
         print("Stack is Empty")
      else:
         print("Topest Element:",temp.data)

   def Display(self):
      temp = self.top
      if temp is None:
         print("Stack is Empty")
      else:
         while temp is not None:
            print("|", temp.data,"|", end="\n")
            temp = temp.link

StackObj = Stack()
while True:
   print("\n****Operations On Stack ***")
   print("1.Push")
   print("2.PoP")
   print("3.Peek")
```

```
        print("4.Display")
        print("5.Exit")
        choice = int(input("Enter Your Choice:"))
        if choice == 1:
            StackObj.Push()
        elif choice == 2:
            StackObj.Pop()
        elif choice == 3:
            StackObj.Peek()
        elif choice == 4:
            StackObj.Display()
        elif choice == 5:
            exit(0)
        else:
    print("Invalid Choice! Try Again:")
```

Queue Program using list:

```
class Queue:
    def __init__(self):
        self.Q = []
        self.front = int(-1)
        self.rear = int(-1)
        self.max = int(3)

    def Enqueue(self):
        if self.front>self.rear:
            self.front = int(-1)
            self.rear = int(-1)
            self.Q = []
        if self.rear == int((self.max-1)):
            print("Queue is Full")
        else:
            if self.front == -1:
                self.front = 0
            val = input("Enter an Element into Queue")
            self.rear = self.rear + 1
            self.Q.append(val)
            print(val,"Inserted sucessfully into Queue")


  def Dequeue(self):
   if (self.front==-1 and self.rear== -1)or(self.front>self.rear):
            self.front=self.rear=-1
     self.Q = []
     print("Queue is Empty")
   else:
        val = self.Q[self.front]
        self.Q[self.front]=""
        self.front = self.front+1
        print(val," is deleted Sucesfully")

   def Display(self):
    if (self.front==-1 and self.rear==-1)or(self.front>self.rear):
            self.front=self.rear=-1
            print("Queue is Empty")
            self.Q = []
   else:
            for i in range(self.front,self.rear+1):
                print(self.Q[i],"<--",end="")
             Qobj = Queue()

             while True:
                 print("\n****Operations On Stack ***")
```

```
print("1.Enqueue")
print("2.Dequeue")
print("3.Display")
print("4.Exit")
choice = int(input("Enter Your Choice:"))
if choice == 1:
    Qobj.Enqueue()
elif choice == 2:
    Qobj.Dequeue()
elif choice == 3:
    Qobj.Display()
elif choice == 4:
    exit(0)
else:
    print("Invalid Choice! Try Again:")
```

Queue Program using linked list:

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

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

    def Enqueue(self):
        temp = self.front
        data_in=input("Enter an Element into Queue")
        NewNode = Node(data_in)
        if temp is None:
            self.front = NewNode
            self.rear = NewNode
        else:
            while temp.link is not None:
                temp = temp.link
            temp.link = NewNode
            self.rear = NewNode
        print(data_in, " Inserted sucessfully into Queue")

    def Dequeue(self):
        temp = self.front
        if temp is not None:
            self.front = temp.link
            print(temp.data, "is deleted Sucesfully")
            if self.front is self.rear:
                self.front=self.rear=None
            temp = None
        else:
            print("Queue is Empty")

    def Display(self):
        temp = self.front
        if temp is None:
            print("Queue is Empty")
        else:
            while temp is not None:
                print(temp.data,"<--", end="")
                temp = temp.link
Qobj = Queue()

while True:
```

```
print("\n****Operations On Stack ***")
print("1.Enqueue")
print("2.Dequeue")
print("3.Display")
print("4.Exit")
choice = int(input("Enter Your Choice:"))
if choice == 1:
   Qobj.Enqueue()
elif choice == 2:
   Qobj.Dequeue()
elif choice == 3:
   Qobj.Display()
elif choice == 4:
   exit(0)
else:
   print("Invalid Choice! Try Again:")
```

Output:

10.Write a program to implement Singly Linked List

Program:

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

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

   def InsertAtBeg(self, data_in):
      NewNode = Node(data_in)
      NewNode.link = self.head
      self.head = NewNode
      self.TraverseList()

   def InsertAtEnd(self, data_in):
      temp = self.head
      NewNode = Node(data_in)
      if temp is None:
         #print("List Empty")
         self.head=NewNode
      else:
         while temp.link is not None:
            temp = temp.link
         temp.link = NewNode
      self.TraverseList()

   # Function to remove node1
   def RemoveNodeAtBeg(self):
      temp = self.head
      if temp is not None:
            self.head = temp.link
            print(temp.data,"is Deleted from List")
            temp = None
      else:
         print("List is Empty")

   def RemoveNodeAtEnd(self):
      temp = self.head
      if temp==None:
         print("List Empty")
      elif temp.link == None:
```

```
            print(temp.data, "is Deleted from List")
            temp = None
            self.head=None
        else:
                while temp.link is not None:
                    prev = temp
                    temp = temp.link
                print(temp.data,"is Deleted from List")
                prev.link = None
                temp=None
        self.TraverseList()

    def TraverseList(self):
        temp = self.head
        if temp is None:
            print("Linked List is Empty")
        else:
            while temp is not None:
                print("-->", temp.data, end="")
                temp = temp.link


    def NodeCount(self):
        count = 0
        temp = self.head
        if temp is None:
            return count
        else:
            while temp is not None:
                count = count+1
                temp = temp.link
            return count

    def InsertAtPos(self, data_in,pos):
        NewNode = Node(data_in)
        Nc = int(self.NodeCount())
        if(pos > Nc and Nc == 0)or(pos > Nc and Nc != 0):
            if pos == 1:
                NewNode.link = self.head
                self.head = NewNode
            else:
                print("Invalid Position\nTry Again")
        else:
            cur = self.head
            prev = cur
            count = int(1)
            while count < pos:
                print("While")
```

```
                prev = cur
                cur = cur.link
                count = count+1
            if pos==1:
                NewNode.link = self.head
                self.head = NewNode
            else:
                NewNode.link = cur
                prev.link = NewNode
        self.TraverseList()

#Delete a node at a position
    def DelAtPos(self, pos):
        Nc = int(self.NodeCount())
        if pos > Nc:
            print("Invalid Position\nTry Again")
        elif Nc == 0:
            print("List is empty")
            print("Deletion Not Possible")
        elif pos == 1:
            temp = self.head
            print(temp.data, " is Deleted from List")
            temp = temp.link
            self.head = temp
        else:
            cur = self.head
            count = int(1)
            while count < pos:
                prev = cur
                cur = cur.link
                count = count+1
            temp = cur
            prev.link=cur.link
            print(temp.data, " is Deleted from List")
            temp = None
#Search
    def search(self, key):
        count = 1
        Loc = int(-1)
        temp = self.head
        if temp is None:
             print("List is Empty")
        else:
            while temp is not None:
                if key == int(temp.data):
                    Loc = count
                    break
                count = count+1
```

```
            temp = temp.link
        if Loc != -1:
            print(key ,"Found at Location ",Loc)
        else:
            print(key, "NOt Found in the List ")

#Creating Object to List ADT
llist = SLinkedList()

while True:
    print("\n****Operations On Single Linked List***")
    print("1.Insert at Begining")
    print("2.Insert at End")
    print("3.Delete at Begining")
    print("4.Delete at End")
    print("5.Traverse the List")
    print("6.Node Count")
    print("7.Insert at a Position")
    print("8.Delete at a Position")
    print("9.Search for a Node")
    print("10.Exit")
    choice = int(input("Enter Your Choice:"))
    if choice == 1:
        data = input("Enter a Value:")
        llist.InsertAtBeg(data)
    elif choice == 2:
        data = input("Enter a Value:")
        llist.InsertAtEnd(data)
    elif choice == 3:
        llist.RemoveNodeAtBeg()
    elif choice == 4:
        llist.RemoveNodeAtEnd()
    elif choice == 5:
        llist.TraverseList()
    elif choice == 6:
        print("Total nodes in the List:",llist.NodeCount())
    elif choice == 7:
        data = input("Enter a Value:")
        nCount=int(llist.NodeCount())
        if nCount==0:
            print("Empty List:")
        else:
            print("Available max position is ",nCount )
        pos = int(input("Enter position of insertion:"))
        llist.InsertAtPos(data, pos)
    elif choice == 8:
        pos = int(input("Enter position for Deletion:"))
        llist.DelAtPos(pos)
```

```
        elif choice == 9:
            keyVal = int(input("Enter key  for searching:"))
            llist.search(keyVal)
        elif choice == 10:
            exit(0)
        else:
print("Invalid Choice! Try Again:")
```

11.Write a program to implement Doubly Linked list
Program:

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

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

   def InsertAtBeg(self, data_in):
      NewNode = Node(data_in)
      NewNode.next = self.head
      self.head = NewNode
      self.DisplayList()

   def InsertAtEnd(self, data_in):
      temp = self.head
      NewNode = Node(data_in)
      if temp is None:
         self.head=NewNode
      else:
         while temp.next is not None:
            temp = temp.next
         temp.next = NewNode
         NewNode.prev = temp
      self.DisplayList()

   # Function to remove node
   def RemoveNodeAtBeg(self):
      temp = self.head
      if temp.next is None:
         print(temp.data, "is Deleted from List")
         temp = None
         self.head=temp
      elif temp is not None:
         print(temp.data, "is Deleted from List")
         temp = temp.next
         temp.prev=None
         self.head=temp
         temp = None
      else:
         print("List is Empty")
      self.DisplayList()

   def RemoveNodeAtEnd(self):
```

```
        temp = self.head
        if temp==None:
            print("List is Empty")
        elif temp.next == None:
            print(temp.data, "is Deleted from List")
            temp = None
            self.head=None
        else:
                while temp.next is not None:
                    pr = temp
                    temp = temp.next
                print(temp.data,"is Deleted from List")
                temp = None
                pr.next = None
        self.DisplayList()

    def DisplayList(self):
        temp = self.head
        if temp is None:
            print("Doubly Linked List is Empty")
        else:
            while temp is not None:
                print("<==>", temp.data, end="")
                temp = temp.next
        print()

    def NodeCount(self):
        count = 0
        temp = self.head
        if temp is None:
            return count
        else:
            while temp is not None:
                count = count+1
                temp = temp.next
            return count

    def InsertAtPos(self, data_in,pos):
        NewNode = Node(data_in)
        Nc = int(self.NodeCount())
        if(pos > Nc and Nc == 0)or(pos > Nc and Nc != 0):
            if pos == 1:
                NewNode.next = self.head
                self.head = NewNode
            else:
                print("Invalid Position\nTry Again")
        else:
            cr = self.head
            pr = cr
```

```
            count = int(1)
            while count < pos:
                print("While")
                pr = cr
                cr = cr.next
                count = count+1
            if pos==1:
                NewNode.next = self.head
                self.head = NewNode
            else:
                NewNode.next = cr
                cr.prev = NewNode
                pr.next = NewNode
                NewNode.prev = pr
        self.DisplayList()

#Delete a node at a position
    def DelAtPos(self, pos):

        Nc = int(self.NodeCount())
        if pos > Nc:
            print("Invalid Position\nTry Again")
        elif Nc == 0 :
            print("List is empty")
        elif pos == 1:
            temp = self.head
            print(temp.data, "is Deleted from List")
            temp = temp.next
            self.head = temp
            if temp is None:
                pass
            else:
                temp.prev = None
                temp = None
        elif Nc==pos:
            temp = self.head
            while temp.next is not None:
                pr = temp
                temp = temp.next
            print(temp.data, "is Deleted from List")
            temp = None
            pr.next = None
        else:
            cr = self.head
            pr = cr
            count = int(1)
            while count < pos:
                pr = cr
                cr = cr.next
```

```
                count = count+1
            Dnode=cr
            print(Dnode.data, " is Deleted from List")
            Dnode = None
            pr.next = cr.next
            temp = cr.next
            temp.prev = pr
        self.DisplayList()

#Search
    def search(self, key):
        count = 1
        Loc = int(-1)
        temp = self.head
        if temp is None:
             print("List is Empty")
        else:
            while temp is not None:
                if key == int(temp.data):
                    Loc = count
                count = count+1
                temp = temp.next

        if Loc != -1:
            print(key ,"Found at Location ",Loc)
        else:
            print(key, "NOt Found in the List ")

#Creating Object to List ADT
dll = DLinkedList()
while True:
    print("****Operations On Doubly Linked List***")
    print("1.Insert at Begining")
    print("2.Insert at End")
    print("3.Delete at Begining")
    print("4.Delete at End")
    print("5.Display")
    print("6.Node Count")
    print("7.Insert at a Position")
    print("8.Delete at a Position")
    print("9.Search for a Node")
    print("10.Exit")
    choice = int(input("Enter Your Choice:"))
    if choice == 1:
        data = input("Enter a Value:")
        dll.InsertAtBeg(data)
    elif choice == 2:
        data = input("Enter a Value:")
        dll.InsertAtEnd(data)
```

```
        elif choice == 3:
            dll.RemoveNodeAtBeg()
        elif choice == 4:
            dll.RemoveNodeAtEnd()
        elif choice == 5:
            dll.DisplayList()
        elif choice == 6:
            print("Total nodes in the List:",dll.NodeCount())
        elif choice == 7:
            data = input("Enter a Value:")
            nCount=int(dll.NodeCount())
            if nCount==0:
                print("Empty List:")
            else:
                print("Available max position is ",nCount )
            pos = int(input("Enter position of insertion:"))
            dll.InsertAtPos(data, pos)
        elif choice == 8:
            pos = int(input("Enter position for Deletion:"))
            dll.DelAtPos(pos)
        elif choice == 9:
            keyVal = int(input("Enter key  for searching:"))
            dll.search(keyVal)
        elif choice == 10:
            exit(0)
        else:
    print("Invalid Choice! Try Again:")
```

12.Write a program to implement Binary Search Tree

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

 def insert(self, val):
    if not self.val:
       self.val = val
       return

    if self.val == val:
       return

    if val < self.val:
       if self.left:
          self.left.insert(val)
          return
       self.left = BSTNode(val)
       return

    if self.right:
       self.right.insert(val)
       return
    self.right = BSTNode(val)

 def get_min(self):
    current = self
    while current.left is not None:
       current = current.left
    return current.val

 def get_max(self):
    current = self
    while current.right is not None:
       current = current.right
    return current.val

 def delete(self, val):
    if self == None:
       return self
    if val < self.val:
       if self.left:
          self.left = self.left.delete(val)
       return self
    if val > self.val:
       if self.right:
```

```
            self.right = self.right.delete(val)
        return self
    if self.right == None:
        return self.left
    if self.left == None:
        return self.right
    min_larger_node = self.right
    while min_larger_node.left:
        min_larger_node = min_larger_node.left
    self.val = min_larger_node.val
    self.right = self.right.delete(min_larger_node.val)
    return self

def exists(self, val):
    if val == self.val:
        return True

    if val < self.val:
        if self.left == None:
            return False
        return self.left.exists(val)

    if self.right == None:
        return False
    return self.right.exists(val)

def preorder(self, vals):
    if self.val is not None:
        vals.append(self.val)
    if self.left is not None:
        self.left.preorder(vals)
    if self.right is not None:
        self.right.preorder(vals)
    return vals

def inorder(self, vals):
    if self.left is not None:
        self.left.inorder(vals)
    if self.val is not None:
        vals.append(self.val)
    if self.right is not None:
        self.right.inorder(vals)
    return vals

def postorder(self, vals):
    if self.left is not None:
        self.left.postorder(vals)
    if self.right is not None:
        self.right.postorder(vals)
```

```
        if self.val is not None:
            vals.append(self.val)
        return vals

nums = [12, 6, 18, 19, 21, 11, 3, 5, 4, 24, 17]
bst = BSTNode()
for num in nums:
    bst.insert(num)
print("preorder:")
print(bst.preorder([]))
print("#")

print("postorder:")
print(bst.postorder([]))
print("#")

print("inorder:")
print(bst.inorder([]))
print("#")

nums = [2, 6, 20]
print("deleting " + str(nums))
for num in nums:
    bst.delete(num)
print("#")

print("4 exists:")
print(bst.exists(4))
print("2 exists:")
print(bst.exists(2))
print("12 exists:")
print(bst.exists(12))
print("18 exists:")
print(bst.exists(18))
```