

Week -10

Q1

AIM: To WAP to implement Queues using arrays.

Description: A Queue is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO). Queues using arrays can be implemented using lists in python. We can use the append() and pop() methods to add or remove the elements respectively. We can define an attribute size to impose Overflow and Underflow conditions.

Code:

```
class Queue:
```

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

```
        self.Q = []
```

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

```
        self.Q.append(data)
```

```
    def Pop(self):
```

```
        if not len(self.Q):
```

```
            print("List is empty")
```

```
        else:
```

```
            self.Q.pop(0)
```

```
    def Display(self):
```

```
        print(*self.Q)
```

```
Q = Queue()
```

```
while True:
```

```
    print("-----Menu-----\n1. Insert 2. Pop 3. Display\n4. Exit\n")
```

```
    choice = int(input("Enter Option: "))
```

```
    if choice == 1:
```

```
        Q.Insert(int(input("Enter data: ")))
```

```
    elif choice == 2:
```

```
    Q.Pop()  
elif choice == 3:  
    Q.Display()  
else:  
    exit()
```

Output:

```
-----Menu-----  
1. Insert 2. Pop 3. Display  
4. Exit  
  
Enter Option: 1  
Enter data: 36  
-----Menu-----  
1. Insert 2. Pop 3. Display  
4. Exit  
  
Enter Option: 3  
52 36  
-----Menu-----  
1. Insert 2. Pop 3. Display  
4. Exit  
  
Enter Option: 2  
-----Menu-----  
1. Insert 2. Pop 3. Display  
4. Exit  
  
Enter Option: 3  
36  
-----Menu-----  
1. Insert 2. Pop 3. Display  
4. Exit  
  
Enter Option: 4
```

Result: The code is error free and runs as expected.

Q2

AIM: To WAP to implement Queue using linked list.

Description: A Queue is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO). Queues can be implemented using Linked lists. Class Node is declared with necessary attributes – ‘next’, ‘data’ which helps to define the structure of every node created. Linked list contains the methods that we want to perform on the Queue.

Code:

```
class Node:
```

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

```
class Queue:
```

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

```
    def Enqueue(self):
```

```
        key=int(input("Enter the value: "))  
        newnode=Node(key)  
        if self.head is None:  
            self.head=newnode  
            self.tail=self.head
```

```
        else:
```

```
            self.tail.next=newnode  
            self.tail=newnode
```

```
    def Dequeue(self):
```

```
        if self.head is None:  
            print("The queue is empty")
```

```
    else:
        self.head=self.head.next
def Display(self):
    if self.head is None:
        print("The queue is empty")
    else:
        ptr=self.head
        while ptr!=None:
            print(ptr.data,end=" ")
            ptr=ptr.next
q=Queue()
while True:

n=int(input("***MENU***\n1.Insertion\t2.Deletion\t3.Display\t4.Exit\nEnter
your choice: "))

    if n==1:
        q.Enqueue()
    elif n==2:
        q.Dequeue()
    elif n==3:
        q.Display()
    elif n==4:
        exit()
    else:
        print("Wrong choice, try again")
```

Output:

```
***MENU***
1.Insertion      2.Deletion      3.Display      4.Exit
Enter your choice: 1
Enter the value: 52
***MENU***
1.Insertion      2.Deletion      3.Display      4.Exit
Enter your choice: 1
Enter the value: 65
***MENU***
1.Insertion      2.Deletion      3.Display      4.Exit
Enter your choice: 3
52 65 ***MENU***
1.Insertion      2.Deletion      3.Display      4.Exit
Enter your choice: 2
***MENU***
1.Insertion      2.Deletion      3.Display      4.Exit
Enter your choice: 3
65 ***MENU***
1.Insertion      2.Deletion      3.Display      4.Exit
Enter your choice: 4
```

Result: The code is error free and runs as expected.

Q3

AIM: To WAP to implement Merge Sort.

Description: Merge Sort is a Divide and Conquer algorithm. It divides the input array into two halves, calls itself for the two halves, and then merges the two sorted halves. The merge() function is used for merging two halves. The merge(arr, l, m, r) is a key process that assumes that arr[l..m] and arr[m+1..r] are sorted and merges the two sorted sub-arrays into one.

Code:

```
def Merge(a,low,mid,high):
```

```
    c=list()
```

```
    i=low
```

```
    j=mid+1
```

```
    while (i<=mid and j<=high):
```

```
        if a[i]<a[j]:
```

```
            c.append(a[i])
```

```
            i+=1
```

```
        else:
```

```
            c.append(a[j])
```

```
            j+=1
```

```
    while i<=mid:
```

```
        c.append(a[i])
```

```
        i+=1
```

```
    while j<=high:
```

```
        c.append(a[j])
```

```
        j+=1
```

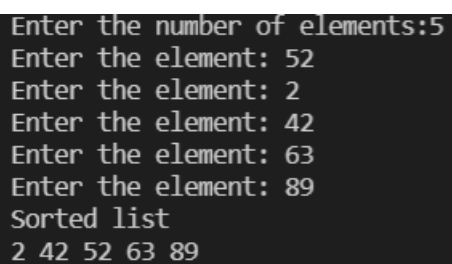
```
    for i in range(len(c)):
```

```
        a[low+i]=c[i]
```

```
def MergeSort(list,low,high):
```

```
mid=(low+high)//2
if (low<high):
    MergeSort(list,low,mid)
    MergeSort(list,mid+1,high)
    Merge(list,low,mid,high)
return list

l=[]
n=int(input("Enter the number of elements:"))
for i in range(n):
    y=int(input("Enter the element: "))
    l.append(y)
x=len(l)
MergeSort(l,0,x-1)
print("Sorted list")
print(*l)
```

Output:A screenshot of a terminal window showing the execution of the Merge Sort program. The user enters 5 for the number of elements, followed by five elements: 52, 2, 42, 63, and 89. The program then prints the sorted list: 2 42 52 63 89.

```
Enter the number of elements:5
Enter the element: 52
Enter the element: 2
Enter the element: 42
Enter the element: 63
Enter the element: 89
Sorted list
2 42 52 63 89
```

Result: The code is error free and runs as expected.