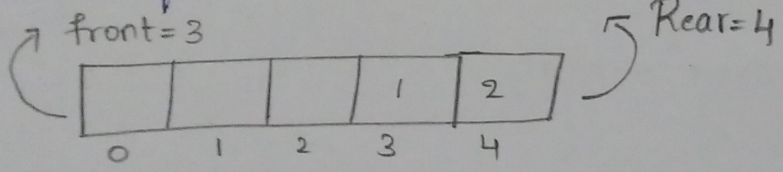
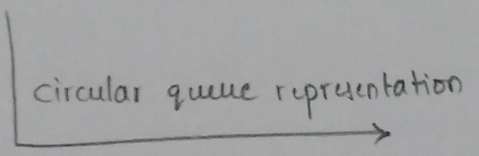
7 19131A0596

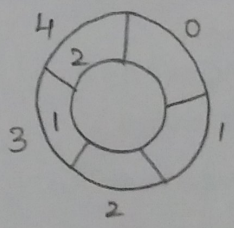
Aim:- Implementation of Circular queue.

Theory:

There was one limitation in the array implementation of queue. if the rear reaches the end position of the queue then there might be possibility of some vacant places available at the beginning cannot be utilized, to overcome this circular queue is introduced.




A circular queue is also a linear data structure, which follows the principle of FIFO (first in first out), but instead of ending the queue at the last position, it again starts

program:

```
#include<iostream>
using namespace std;
void enqueue();
void dequeue();
void display();
int front=-1, rear=-1;
int queue[5], n=5;
int main()
{
   int choice;
   cout<<" 1. enqueue operation \n 2. Dequeue operation \n3. display
   the queue \n 4. Exit"<<endl;
  while(1)
  {
   cout<<"enter your choice"<<endl;
   cin>>choice;
   Switch(choice)
   {
    case 1: enqueue();
            break;
    case 2: dequeue();
            break;
    case 3: display();
            break;
    case 4: exit(0);
            break;
    default: cout<<"Enter valid choice"<<endl;
   }
 }
 return 0;
}
```

```
void enqueue()
{
    int val;
    cout<<"Enter the element";
    cin>>val;
    if(front==0 && rear==n-1)
    {
       cout<<"overflow occured"<<endl;
       return;
    }
    else if(front==-1 && rear==-1)
    {
       front=0;
       rear=0;
    }
    else if(rear=n-1 && front!=0)
    {
      rear=0;
    }
    else
      {
        rear=rear+1;
      }
    cout<<"value entered"<<endl;
    queue[rear]=val;
}
void dequeue()
{
    if(front==-1)
    {
      cout<<"under flow occured"<<endl;
```

```
    return;
  }
  else if(front==rear)
  {
     front=-1;
     rear=-1;
  }
  else if(front==n-1)
  {  front=0;
  }
  else
    front=front+1;
}
void display()
 {
   int i;
   if(front==-1)
   cout<<"circular queue is empty!!"<<endl;
  else
   {
      i=front;
     cout<<"circular queue elements are :"<<endl;
     if(front<=rear)
      {
       while(i<=rear)
         cout<<queue[i++]<<endl;
      }
     else
       {
        while(i<=n-1)
         cout<<queue[i++]<<endl;
        i=0;
       while(i<=rear)
      cout<<queue[i++]<<endl;
       }
   }
```

Sample output:-

```
1. enqueue operation
2. Dequeue operation
3. Display the queue
4. Exit
Enter your choice
1
Enter the element 45
Value entered
Enter your choice
1
Enter the element 78
Value entered
Enter your choice
1
Enter the element 95
value entered
Enter your choice
3
Circular queue Elements are:
45
78
95
Enter your choice
2
Enter your choice
3
circular queue Elements are:
78
95
```

Aim:- Implementation of priority queue

Theory:

A priority queue is an extension of the queue, with some additional property that the elements are arranged on the basis of the priority order.

We can perform insertion and deletion operations on priority queue.

→ These are used to find maximum and minimum values of the given list

program:

```
#include <bits\stdc++.h>
using namespace std;
#define Max 5
void insert(int);
void delete(int);
void create();
void check(int);
void display();
int pri_que[Max];
int front,rear;
int main()
{
   int n,ch;
   Cout<< "\n1- Insert an element into queue";
   Cout<< "\n2- Delete an element from queue";
   Cout<< "\n3- Display queue elements";
   cout << "\n 4- Exit";

   create();
  while(1)
  {
   cout <<"\n Enter your choice : ";
   Cin >> ch;
   Switch(ch)
   {
   case 1:
        cout<<"\n Enter Value to be inserted :";
        Cin>>n;
        insert(n);
        break;
  Case 2:
        Cout << "\n Enter Value to delete :";
        Cin >>n;
        delete(n);
        break;
```

```
case 3:
        display();
        break;
case 4:
        exit(0);
default:
     cout<<"\n choice is incorrect";
  }
 }
  return 0;
}
void create()
{
   front=rear=-1;
}
void insert(int data)
{
  if(rear>=Max-1)
   {
    cout<<"\n Queue overflow occured";
     return;
    }
    if((front==-1) && (rear==-1))
    {
      front++;
      rear++;
      pri_que[rear]=data;
      return;
     }
     else
        check(data);
     rear++;
  }
```

```
Void check(int data)
{
   int i,j;
  for(i=0;i<=rear;i++)
   {
     if(data>=pri_que[i])
      {
        for(j=rear+1;j>i;j--)
        {
          pri_que[i]=pri_que[j-1];
        }
        pri_que[i]=data;
        return;
      }
    }
    pri_que[i]=data;
}
void delete(int data)
{ int i;
if((front==-1)&&(rear==-1))
{  cout<<"\n Queue underflow occured";
   return;
}
for(i=0;i<=rear;i++)
  {  if(data==pri_que[i])
       {
         for(;i<rear;i++)
           {
             pri_que[i]=pri_que[i+1];
           }
        pri_que[i]=-99;
        rear--;
    if(rear==-1)
        front=-1;
```

```
        return;
      }
    }
    cout<<"\n element not found to delete";
}
void display()
{
  if((front==-1)&&(rear==-1))
    {
      cout<<"\n Queue is empty";
      return;
    }
  for( ; front<=rear; front++)
    {
      cout<<pri_que[front]<<endl;
    }
  front=0;
}
```

Sample output:

```
1- Insert an element into queue
2- Delete an element from queue
3- Display queue elements
4- Exit

Enter your choice : 1
enter value to be inserted : 45
Enter your choice : 2
enter value to delete: 45
enter your choice : 2
Queue underflow occured

enter your choice : 3
Queue is empty

enter your choice : 1
enter value to be
  inserted : 67
enter your choice : 3
67
```