Submitted By: Ravi Pandey(075BCT065)

1)Perform enqueue and dequeue operations in linear queue

ALGORITHM:

Enqueue:

1. Get the item to insert to the queue from user
2. If rear index == sizeofarray-1 then print overflow message or terminate the program
3. Else Increase the rear index by 1
4. Then insert the item at rear index of the array.

Dequeue:

1. Check if the array is empty i.e front index==-1.If so, print underflow or terminate the program
2. If all items have been dequeued i.e. front index==sizeofarray-1 return underflow or terminate
3. Else If array has one or more items then set temp=queue[front] and increse front index by 1
4. Return temp as dequeued item to caller.

PROGRAM CODE:

```
/*WAP to Perform enqueue and dequeue operations in Linear Queue*/
#include <iostream>
using namespace std;
template<class t>
class Queue
{
  int front;
  int rear;
  const int SIZE;
  t * q;

  public:
  Queue(int n):SIZE(n)
  {
      q=new t[SIZE];
      front=-1;
      rear=-1;
  }
  ~Queue()
  {
      delete []q;
  }
  class OVERFLOW{};                          /*Exception for queue is full*/
  class UNDERFLOW{};                         /*Exception for queue is empty*/
  bool isFull()
```

```
  {
      if(rear==SIZE-1)
      {
          throw(OVERFLOW());
          return true;
      }
      return false;
  }
  bool isEmpty()
  {
      if(rear==-1 || front==SIZE-1) /*front does not go back after
                                reaching the end even if space is available*/
      {
          throw(UNDERFLOW());
          return true;
      }
      return false;
  }
  void enqueue(t item)
  {
      if(rear==-1)
      front=0;
              rear++;                 /*points rear towards next available space*/
              q[rear]=item;           /*assigns passed data to that location*/
  }
  t dequeue()
  {
      t data;
          data=q[front];     /*returns data pointed by front(Not necessary to do
                               while performing dequeue operation)*/
          front++;            /*points front towards next element of queue*/
      return data;
  }
  void show()
  {
      cout<<"Queue:\t";
      if(front==SIZE-1)
      cout<<q[front]<<"\t";
      else if(!isEmpty())
      for(int i = front;i<=rear;i++)
      cout<<q[i]<<"\t";
  }
};
int main()
{
```

```
    int sizeQ;
    cout<<"Enter Queue Size: ";
    cin>>sizeQ;
    Queue<int> a(sizeQ);
    char op;
    int sel,n;
    do
    {
    try
    {
    cout<<"Enter operation:\n1)Enqueue\n2)Dequeue:\t";
    cin>>sel;
    if(sel==1 && !a.isFull())
    {
        cout<<"Enter data: ";
        cin>>n;
        a.enqueue(n);
        cout<<n<<" enqueued\n";
    }
    else if(sel==2 && !a.isEmpty())
    {
        cout<<a.dequeue()<<" dequeued\n";
    }
    else
    {
        cout<<"Invalid choice!";
    }
    a.show();
    }
    catch(Queue<int>::OVERFLOW)
    {
        cout<<"Queue is full!";
        op=='Y';
    }
    catch(Queue<int>::UNDERFLOW)
    {
        cout<<"Queue is empty!";
        op=='Y';
    }
    cout<<"\nContinue?:";
    cin>>op;
    }while(toupper(op)=='Y' || op=='1');
    return 0;
}
```

OUTPUT:

```
Enter Queue Size: 3
Enter operation:                1)Enqueue       2)Dequeue:      1
Enter data: 5
5 enqueued
Queue:  5
Continue?:y
Enter operation:                1)Enqueue       2)Dequeue:      1
Enter data: 6
6 enqueued
Queue:  5       6
Continue?:y
Enter operation:                1)Enqueue       2)Dequeue:      1
Enter data: 7
7 enqueued
Queue:  5       6       7
Continue?:y
Enter operation:                1)Enqueue       2)Dequeue:      2
5 dequeued
Queue:  6       7
Continue?:y
Enter operation:                1)Enqueue       2)Dequeue:      2
6 dequeued
Queue:  7
Continue?:y
Enter operation:                1)Enqueue       2)Dequeue:
2
Queue is empty!
Continue?:y
Enter operation:                1)Enqueue       2)Dequeue:      1
Queue is full!
Continue?:
```

2)Perform enqueue and dequeue operations in Circular queue

ALGORITHM:

Enqueue:

1. Get the item to insert to the queue from user
2. If front == rear == -1 i.e. queue is empty then, set front=rear=0 and set queue[rear] =item
3. Then check if (rear+1) mod SIZE == front i.e. queue is fulll then, print overflow and terminate process.(modular division is done so that rear does nt exceed the SIZE)
4. Else if queue is partially filled then, increase rear by 1 so that it does not exceed SIZE of array (i.e. rear=(rear+1)mod SIZE) and set queue[rear] to item.

Dequeue:

1. Check if the array is empty i.e front index==-1.If so, print underflow or terminate the program
2. If front == rear and rear!=-1 i.e. there is one element remaining in queue then set temp to queue[front] and then set front = rear =-1(queue empty condition)
3. Else when queue is partially filled set temp = queue[front] and then set front=(front+1) mod SIZE
4. Return temp as dequeued item to caller.

PROGRAM CODE:

```
/*WAP to Perform enqueue and dequeue operations in Circular Queue*/
#include <iostream>
using namespace std;
template<class t>
class CQueue
{
  int front;
  int rear;
  int count;
  const int SIZE;
  t * q;

  public:
  CQueue(int n):SIZE(n)
  {
      q=new t[SIZE];
      front=-1;
      rear=-1;
      count=0;
  }
  ~CQueue()
  {
      delete []q;
  }
  class OVERFLOW{};
  class UNDERFLOW{};

bool qCon(int op)
{

    if(((rear+1)%SIZE==front) && op==1)           /*Queue is full*/
    {
        throw(OVERFLOW());
        return false;
    }
```

```
    else if((front==rear && front==-1) && op==2)  /*Queue empty condition*/
      {
          throw (UNDERFLOW());
          return false;
      }
      return true;
}
  void enqueue(t item)
  {
      if(front==rear && rear==-1)       /*Queue empty condition*/
      {
          front=0;
          rear=0;
          q[rear]=item;
          count++;
      }
      else if((rear+1)%SIZE==front)     /*Queue is full*/
      {
          //throw(OVERFLOW());          /*handled by qCon()*/
      }
      else                              /*One or more space available in queue*/
      {
          rear=(rear+1)%SIZE;
          q[rear]=item;
          count++;
      }
  }
   t dequeue()
  {
      t data;
      if(front==rear && front==-1)  /*Queue empty condition*/
      {
          //throw (UNDERFLOW());    /*handled by qCon()*/
      }
      else if( front == rear)   /*All items have been dequeued so front and rear
                                are changed to initial conditions*/
     {
          data=q[front];
          front =-1;
          rear = -1;
      }
      else                          /*Queue has more than 1 item*/
      {
          data=q[front];
          front=(front+1)%SIZE;
```

```
        count--;
    }

    return data;
}
void display()
{
    if(qCon(2))
    {
        int i=front;
        int num=0;
        bool loop=true;
        cout<<"Queue: ";
        while(loop && num<=count)
        {
            if(count==1)
            {
                cout<<q[i]<<"\t";
                loop=false;
            }
            else
            {
          cout<<q[i]<<"\t";
          if(i==rear && i==0)
          {
              loop=false;
          }
          else if(i==SIZE-1 && i==rear)
              {
                  cout<<q[i]<<"\t";
                  loop=false;
              }
          else
          {
              i=(i+1)%SIZE;
              if(i==rear)
              {
                  cout<<q[i]<<"\t";
                  loop=false;
              }
          }
            }
          num++;
        }
    }
```

```
  }
};

int main()
{
    int sizeQ;
    cout<<"Enter Queue Size: ";
    cin>>sizeQ;
    CQueue<int> a(sizeQ);
    char op;
    int sel,n;
    do
    {
    try
    {
    cout<<"Enter operation:\t1)Enqueue\t2)Dequeue:\t";
    cin>>sel;
    if(sel==1)
    {
        if(a.qCon(sel))
        {
        cout<<"Enter data: ";
        cin>>n;
        a.enqueue(n);
        cout<<n<<" enqueued.\n";
        }
    }
    else if(sel==2)
    {
        if(a.qCon(sel))
        {
            cout<<a.dequeue()<<" dequed.\n";
        }

    }
    else
    {
        cout<<"Invalid choice!";
    }
    a.display();
    }
    catch(CQueue<int>::OVERFLOW)
    {
        cout<<"Queue is full!";
        op=='Y';
```

```
    }
    catch(CQueue<int>::UNDERFLOW)
    {
        cout<<"Queue is empty!";
        op=='Y';
    }
    cout<<"\nContinue?:";
    cin>>op;
    }while(toupper(op)=='Y' || op=='1');
    return 0;
}
```

OUTPUT:

```
Enter Queue Size: 4
Enter operation:        1)Enqueue       2)Dequeue:      1
Enter data: 5
5 enqueued.
Queue: 5
Continue?:y
Enter operation:        1)Enqueue       2)Dequeue:      1
Enter data: 6
6 enqueued.
Queue: 5        6
Continue?:y
Enter operation:        1)Enqueue       2)Dequeue:      2
5 dequed.
Queue: 6
Continue?:y
Enter operation:        1)Enqueue       2)Dequeue:      1
Enter data: 7
7 enqueued.
Queue: 6        7
Continue?:y
Enter operation:        1)Enqueue       2)Dequeue:      1
Enter data: 8
8 enqueued.
Queue: 6        7       8
Continue?:y
Enter operation:        1)Enqueue       2)Dequeue:      1
Enter data: 9
9 enqueued.
Queue: 6        7       8       9
Continue?:y
Enter operation:        1)Enqueue       2)Dequeue:      1
Queue is full!
Continue?:y
Enter operation:        1)Enqueue       2)Dequeue:      2
6 dequed.
Queue: 7        8       9
Continue?:y
Enter operation:        1)Enqueue       2)Dequeue:      2
7 dequed.
Queue: 8        9
Continue?:y
Enter operation:        1)Enqueue       2)Dequeue:      2
8 dequed.
Queue: 9
```

```
Continue?:y
Enter operation:        1)Enqueue       2)Dequeue:      2
9 dequed.
Queue is empty!
Continue?:y
Enter operation:        1)Enqueue       2)Dequeue:      1
Enter data: 10
10 enqueued.
Queue: 10
Continue?:y
Enter operation:        1)Enqueue       2)Dequeue:      1
Enter data: 11
11 enqueued.
Queue: 10       11
Continue?:
```

3)Perform operations in Deque(Double Ended Queue) for:

a. Add at beginning
b. Add at end
c. Delete from beginning
d. Delete from end

ALGORITHM:

Add at Beginning:

1. Get the item to insert to the deque from user.
2. Check if the deque is empty.If so, set both front and rear to 0 and deque[front]=item.
3. If the deque is full(i.e. (front =0 and rear=SIZE-1) or front=rear+1 then, set print overflow and terminate process.
4. To add at beginning,front is reduced by 1 (front--) except when front=0 ,in which case, front is set to front=SIZE-1(Circular queue approach) and then deque[front] is set to the item provided by user.

Add at End:

1. Get the item to insert to the deque from user.
2. Check if the deque is empty.If so, set both front and rear to 0 and deque[rear]=item.
3. If the deque is full(i.e. (front =0 and rear=SIZE-1) or front=rear+1 then, set print overflow and terminate process.
4. To add at end,rear is increased by 1 (rear++) except when rear=SIZE-1, in which case, rear is set to zero(Circular queue approach) and then deque[rear] is set to the item provided by user.

Delete from Beginning:

1. If the deque is empty(rear==front==-1) print underflow and terminate process.
2. When deque has only one element((front==rear) and front!=-1), set temp=deque[front] and then set both front and rear to -1(front=rear=-1).

3. If front==SIZE-1 then set temp=deque[front] and then set front=0.
4. In all other cases set temp=deque[front] and increase front by 1(front++).
5. Return temp as deleted element to caller.

Delete from End:

1. If the deque is empty(rear==front==-1) print underflow and terminate process.
2. When deque has only one element((front==rear) and front!=-1), set temp=deque[rear] and then set both front and rear to -1(front=rear=-1).
3. If rear==0 then set temp=deque[rear] and then set rear= SIZE-1(Circular queue approach).
4. In all other cases set temp=deque[rear] and decrease rear by 1(rear--).
5. Return temp as deleted element to caller.

PROGRAM CODE:

```
/*WAP to Perform operations in Deque( Double ended queue) for:
a. Add at beginning
b. Add at end
c. Delete from beginning
d. Delete from end*/
#include <iostream>
using namespace std;
template <class t>
class Deque
{
    const int SIZE;
    int front;
    int rear;
    int count;
    t * d;

    public:
    class OVERFLOW{};
    class UNDERFLOW{};
  Deque(int n):SIZE(n)
  {
      d=new t[SIZE];
      front=-1;
      rear=-1;
      count=0;
  }
  ~Deque()
  {
      delete []d;
  }
  void addBegin(t item)
```

```
{
    if((front==0 && rear==SIZE-1) || (front==rear+1)) /*Deque full condition*/
    {
        throw(OVERFLOW());
    }
    else if(front == rear && rear==-1)               /*Initial condition*/
    {
        front =0;
        rear=0;
        d[front]=item;
        count++;
    }
    else if(front ==0)      /*front-- becomes 0 which is not required outcome*/
    {
        front =SIZE-1;
        d[front]=item;
        count++;
    }
    else
    {
        front--;
        d[front]=item;
        count++;
    }
}
void addEnd(t item)                              /*Deque full condition*/
{
    if((front==0 && rear==SIZE-1) || (front==rear+1))
    {
        throw(OVERFLOW());
    }
    else if(front==rear && rear==-1)             /*Initial Condition*/
    {
        front=0;
        rear=0;
        d[rear]=item;
        count++;
    }
    else if(rear==SIZE-1)
    {
        rear=0;
        d[rear]=item;
        count++;
    }
    else
```

```
    {
        rear++;
        d[rear]=item;
        count++;
    }
 }
t delBegin()
 {
    t data;
    if(front==rear && rear==-1)
    {
        throw(UNDERFLOW());
    }
    else if(front == rear)
    {
        data= d[front];
        front=-1;
        rear=-1;
        count--;
    }
    else if(front==SIZE-1)
    {
        data= d[front];
        front=0;
        count--;
    }
    else
    {
        data=d[front];
        front++;
        count--;
    }
    return data;
 }
 t delEnd()
 {
    t data;
    if(front==rear && rear==-1)
    {
        throw(UNDERFLOW());
    }
    else if(front == rear)
    {
        data= d[rear];
        front=-1;
```

```
        rear=-1;
        count--;
    }
    else if(rear==0)
    {
        data= d[rear];
        rear=SIZE-1;
        count--;
    }
    else
    {
        data=d[rear];
        rear--;
        count--;
    }
    return data;
}
void display()
{
    if(front==rear && rear==-1)
    {
        throw(UNDERFLOW());
    }
    else
    {
        int i=front;
        int num=0;
        bool loop=true;
        cout<<"\t\tDeque: ";
        while(loop && num<=count)
        {
            if(count==1)
            {
                cout<<d[i]<<"\t";
                loop=false;
            }
            else
            {
          cout<<d[i]<<"\t";
          if(i==rear && i==0)
          {
              loop=false;
          }
          else if(i==SIZE-1 && i==rear)
              {
```

```
                    cout<<d[i]<<"\t";
                    loop=false;
                }
            else
            {
                i=(i+1)%SIZE;
                if(i==rear)
                {
                    cout<<d[i]<<"\t";
                    loop=false;
                }
            }
              }
            num++;
          }
      }
  }
};
int main()
{
    int sizeQ;
    cout<<"Enter Dueqe Size: ";
    cin>>sizeQ;
    Deque<int> a(sizeQ);
    char op;
    int sel,n;
    do
    {
    try
    {
    cout<<"Enter operation: \n1)Add at Begining\t2)Add at end\t3)Delete from Begi
ning\t4)Delete from End\t";
    cin>>sel;
    switch (sel)
    {
        case (1):
        {
            cout<<"Enter data: ";
            cin>>n;
            a.addBegin(n);
            break;
        }
        case (2):
        {
            cout<<"Enter data: ";
```

```
            cin>>n;
            a.addEnd(n);
            break;
        }
        case(3):
        {
            cout<<a.delBegin()<< " deleted\n";
            break;
        }
        case(4):
        {
            cout<<a.delEnd()<<" deleted\n";
            break;
        }
        default:
        {
            cout<<"Invalid choice!";
        }
    }
    a.display();
    }
    catch(Deque<int>::OVERFLOW)
    {
        cout<<"Deque is full!";
        op=='Y';
    }
    catch(Deque<int>::UNDERFLOW)
    {
        cout<<"Deque is empty!";
        op=='Y';
    }
    cout<<"\nContinue?:";
    cin>>op;
    }while(toupper(op)=='Y' || op=='1');
    return 0;
}
```

OUTPUT:

```
Enter Dueqe Size: 4
Enter operation:
1)Add at Begining       2)Add at end    3)Delete from Begining  4)Delete from End       1
Enter data: 2
               Deque: 2
Continue?:y
Enter operation:
1)Add at Begining       2)Add at end    3)Delete from Begining  4)Delete from End       2
Enter data: 3
               Deque: 2        3
Continue?:y
Enter operation:
1)Add at Begining       2)Add at end    3)Delete from Begining  4)Delete from End       1
Enter data: 1
               Deque: 1        2       3
Continue?:y
Enter operation:
1)Add at Begining       2)Add at end    3)Delete from Begining  4)Delete from End       2
Enter data: 4
               Deque: 1        2       3       4
Continue?:y
Enter operation:
1)Add at Begining       2)Add at end    3)Delete from Begining  4)Delete from End       1
Enter data: 5
Deque is full!
Continue?:y
Enter operation:
1)Add at Begining       2)Add at end    3)Delete from Begining  4)Delete from End       4
4 deleted
               Deque: 1        2       3
Continue?:y
Enter operation:
1)Add at Begining       2)Add at end    3)Delete from Begining  4)Delete from End       3
1 deleted
               Deque: 2        3
Continue?:y
Enter operation:
1)Add at Begining       2)Add at end    3)Delete from Begining  4)Delete from End       4
3 deleted
               Deque: 2
Continue?:y
```

```
Enter operation:
1)Add at Begining       2)Add at end    3)Delete from Begining  4)Delete from End       3
2 deleted
Deque is empty!
Continue?:n
```