|  |  |
| --- | --- |
|  | **DEPARTMENT OF COMPUTER ENGINEERING** |

**Experiment No. 06**

|  |  |
| --- | --- |
| Semester | S.E. Semester III – Computer Engineering |
| Subject | Data Structures Lab (CSL301) |
| Subject Professor In-charge | Prof. Swapnil S. Sonawane |
| Assisting Teachers | Prof. Swapnil S. Sonawane |

|  |  |
| --- | --- |
| Student Name | Trisha Shah |
| Roll Number | 20102A0004 |

**Title:**

Implement Circular Queue ADT using array.

**Objective:**

Students will be able to explain various data structures, related terminologies and its types.

**Explanation:**

Circular Queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle. It is also called ‘Ring Buffer’.

In a normal Queue, we can insert elements until queue becomes full. But once queue becomes full, we can not insert the next element even if there is a space in front of queue.

Operations on Circular Queue:

Front: Get the front item from queue.

Rear: Get the last item from queue.

enQueue(value) This function is used to insert an element into the circular queue. In a circular queue, the new element is always inserted at Rear position.

Check whether queue is Full – Check ((rear == SIZE-1 && front == 0) || (rear == front-1)).

If it is full then display Queue is full. If queue is not full then, check if (rear == SIZE – 1 && front != 0) if it is true then set rear=0 and insert element.

deQueue() This function is used to delete an element from the circular queue. In a circular queue, the element is always deleted from front position.

Check whether queue is Empty means check (front==-1).

If it is empty then display Queue is empty. If queue is not empty then step 3

Check if (front==rear) if it is true then set front=rear= -1 else check if (front==size-1), if it is true then set front=0 and return the element.

**Program Code:**

#include<stdio.h>

#define N 5

typedef struct queue

{

int a[N];

int front,rear,count;

}queue;

void enqueue(queue \*q, int x)

{

if(q->count==N)

printf("\nQueue Overflow...");

else

{

q->count++;

q->rear=(q->rear+1)%N;

q->a[q->rear]=x;

}

}

int isempty(queue \*q)

{

if(q->count==0)

return 1;

else

return 0;

}

int dequeue(queue \*q)

{

int x;

if(isempty(q))

return -1;

else

{

q->count--;

x=q->a[q->front];

q->front=(q->front+1)%N;

return x;

}

}

void display(queue \*q)

{

int i;

if(isempty(q))

{

printf("\nQueue is empty..");

}

else

{

i=q->front;

while(1)

{

printf("\n%d",q->a[i]);

if(i==q->rear)

break;

else

i=(i+1)%N;

}

}

}

int main()

{

int ch,x;

queue q;

q.front=0;

q.rear=-1;

q.count=0;

while(1)

{

printf("\nMenu:\n1-ENQUEUE\n2-DEQUEUE\n3-DISPLAY\n4-EXIT\nEnter Choice=");

scanf("%d",&ch);

if(ch==4)

break;

switch(ch)

{

case 1:

{

printf("\nEnter element to be inserted=");

scanf("%d",&x);

enqueue(&q,x);

}

break;

case 2:

{

x=dequeue(&q);

if(x==-1)

{

printf("\nQueue Underflow...");

}

else

{

printf("\nDeleted Element=%d",x);

}

}

break;

case 3:

{

display(&q);

}

break;

default:

{

printf("\nInvalid Choice...");

}

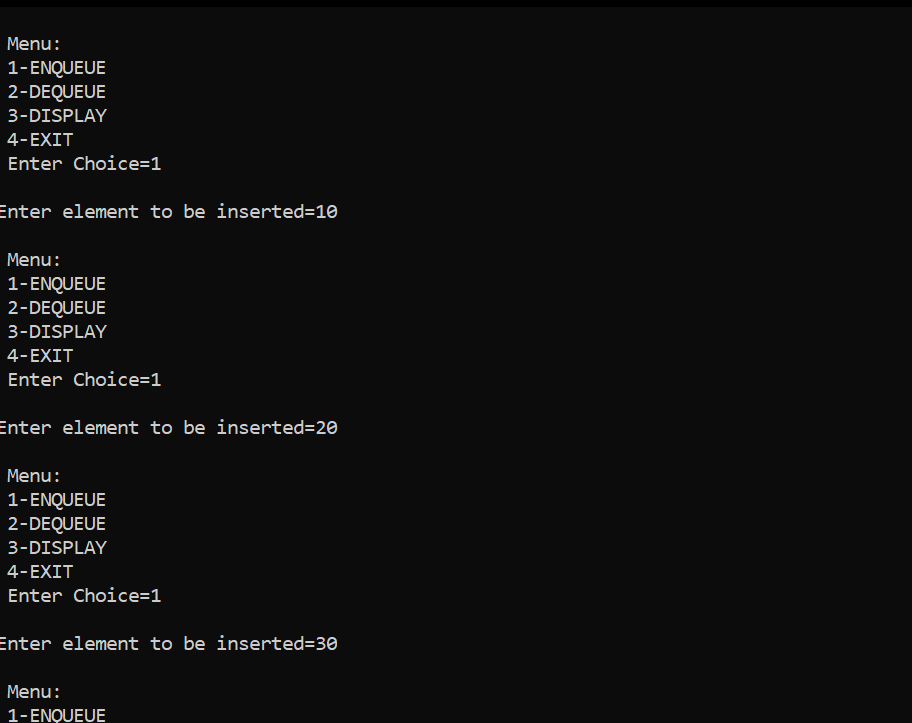
}

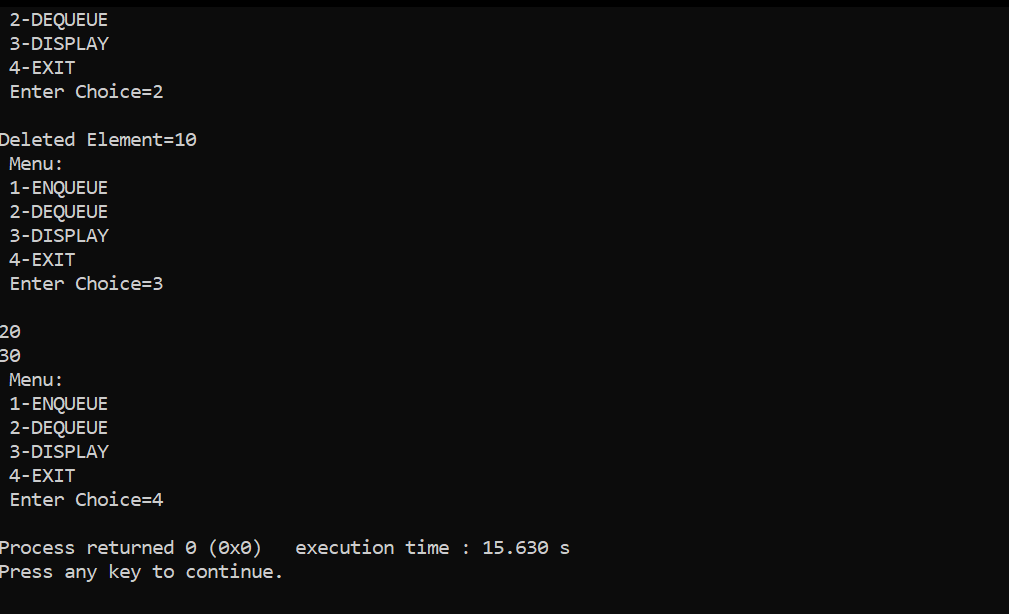
}

return 0;

}

**Output:**

****

****

**Conclusion:**

Through this experiment, students were able to successfully implement a circular queue.