Experiment No.4
Aim: To implement a Linear Queue using arrays
Name: Nikhil Kamalaji Shingade
Roll no: 57
Date of Performance:
Date of Submission:

Experiment No. 4: Simple Queue Operations

Aim: To implement a Linear Queue using arrays.

Objective:

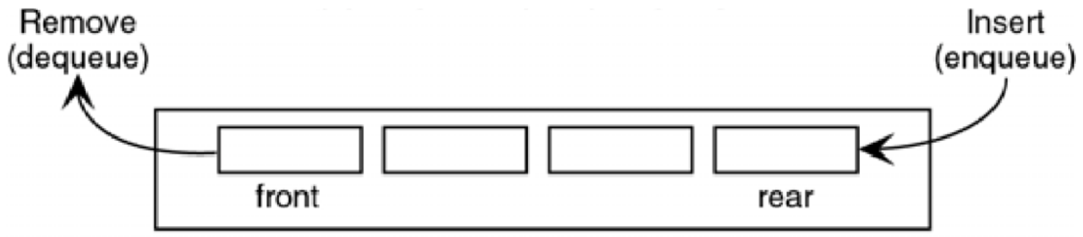
1 Understand the Queue data structure and its basic operations.

2. Understand the method of defining Queue ADT and its basic operations.

3. Learn how to create objects from an ADT and member functions are invoked.

Theory:

A queue is an ordered collection where items are removed from the front and inserted at the rear, following the First-In-First-Out (FIFO) order. The fundamental operations for a queue are "Enqueue," which adds an item to the rear, and "Dequeue," which removes an item from the front.



(b) A computer queue

Typically, a one-dimensional array is used to implement a queue, and two integer values, FRONT and REAR, track the front and rear positions in the array. When an element is removed from the queue, FRONT is incremented by one, and when an element is added to the queue, REAR is increased by one. This ensures that items are processed in the order they were added, maintaining the FIFO principle.

Algorithm:

ENQUEUE(item)

```
1. If (queue is full) Print "overflow"

2. if (First node insertion)
```

```
    Front++
3.  rear++
Queue[rear]=value

DEQUEUE()
1. If (queue is empty)
    Print "underflow"
2. if(front=rear)
        Front=-1 and rear=-1
3.  t = queue[front]
4.  front++
5.  Return t

ISEMPTY()
1. If(front = -1)then return 1
2. return 0

ISFULL()
1. If(rear = max)then return 1
2. return 0
```

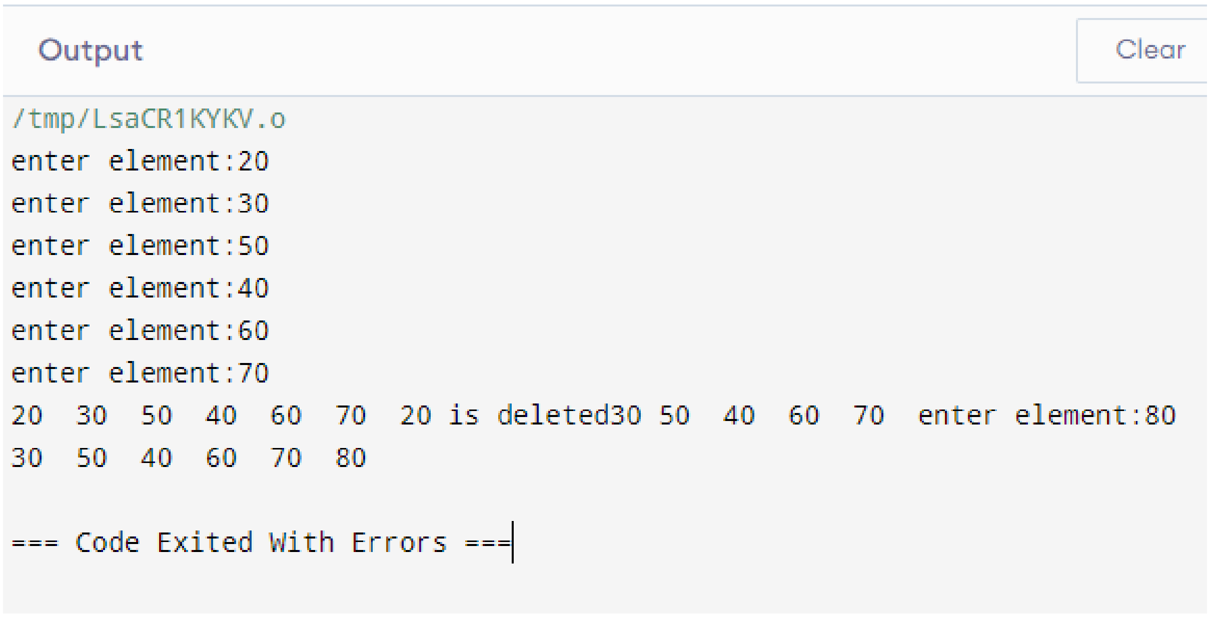
Code:

```c
#include<stdio.h>

#define max 10
int front=-1,rear=-1,a[max],i,num;
void enqueue()
{
```

```
printf("enter element:");
scanf("%d",&num);
if(rear==max-1)
{
printf("queue is full");
}
else
{
if(front==-1)
{
front++;
}
rear=(rear+1);
a[rear]=num;
}
}
void dequeue()
{
if(front==-1 && rear==-1)
{
printf("queue is empty");
}
else
{
num=a[front];
front++;
printf("%d is deleted",num);
}
}
void display()
{
if(front==-1)
{
printf("queue is empty");
}
else
{
for(i=front;i<=rear;i++)
{
printf("%d\t",a[i]);
}
}
}
void main()
{
```

```
enqueue();
enqueue();
enqueue();
enqueue();
enqueue();
enqueue();
display();
dequeue();
display();
enqueue();
display();

}
```

Output:



```
/tmp/LsaCR1KYKV.o
enter element:20
enter element:30
enter element:50
enter element:40
enter element:60
enter element:70
20  30  50  40  60  70  20 is deleted30 50  40  60  70  enter element:80
30  50  40  60  70  80

=== Code Exited With Errors ===
```

Conclusion:

1) What is the structure of queue ADT?

ANS:

Queue Abstract Data Type (ADT) operates on the principle of First In, First Out (FIFO), meaning the first element added is the first one to be removed. Here's the structure:

Basic Operations:

- **Enqueue**: Adds an element to the end of the queue.
- **Dequeue**: Removes and returns the element from the front of the queue.
- **Peek/Front**: Returns the front element without removing it.
- **IsEmpty**: Checks if the queue is empty.
- **IsFull**: (Optional) Checks if the queue is full. This is relevant in bounded queues.

Conceptual Structure:

1. **Front**: Points to the first element.
2. **Rear**: Points to the last element.
3. **Elements**: The data items stored in the queue.
   Queues can be implemented using arrays, linked lists, or circular buffers, depending o n the requirements.

2) List various applications of queues?
ANS:
Queues are foundational in both computing and everyday applications. Here are some not able ones:

- **Task Scheduling**: Operating systems use queues to manage tasks and processes.
- **Print Spooling**: Ensures documents are printed in the order they were sent to the print er.
- **Customer Service**: Manages calls or service requests in the order they arrive.
- **Data Buffering**: Used in streaming audio/video to handle data flow.
- **CPU Scheduling**: Organizes processes based on priority and arrival time.
- **Breadth-First Search (BFS)**: Utilizes queues to explore graph nodes level by level.
- **Network Routers**: Queues help manage data packets to be sent over networks.

3) Where is queue used in a computer system proceesing?
ANS:
Queues are essential in various areas of computer system processing. Some key uses include:

1. Task Scheduling:
- Operating systems use queues to manage the execution of processes, ensuring they are run in the order they arrive.

2. Print Spooling:
- Manages print jobs sent to a printer, ensuring they are printed in the order they were r eceived.

3. Data Buffering:
- Used in network data transfers and streaming services to handle bursts of data and mai ntain a steady flow.

4. CPU Scheduling:
- The CPU uses queues to manage processes waiting for execution, prioritizing based o n arrival and priority.

5. Breadth-First Search (BFS):
- Utilized in graph algorithms to explore nodes level-by-level.

6. I/O Request Management:
- Manages read/write requests to ensure they are processed efficiently and in the right o rder.