Experiment No.4
Implementation of Queue menu driven program using arrays
Name: Sharvari Anand Bhondekar
Roll No: 06
Date of Performance:
Date of Submission:
Marks:
Sign:

Experiment No. 4: Simple Queue Operations

Aim: To implement a Linear Queue using arrays.

Objective:

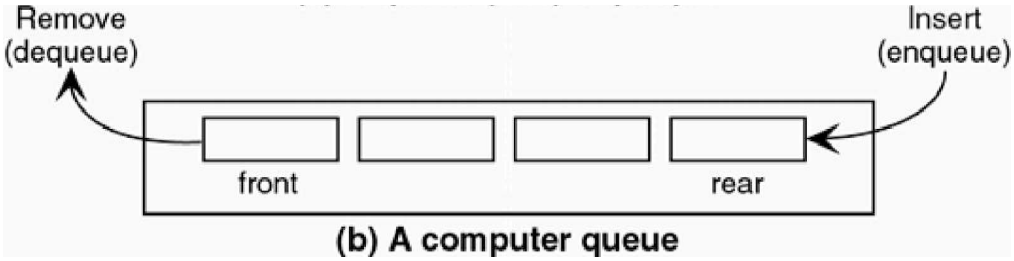
1 Understand the Queue data structure and its basic operations.

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:

```
#include <stdio.h>
#include <ctype.h>
#define MAX 100
int Front = -1, Rear = -1, q[MAX], n;
void Enqueue()
{
printf("Enter a Number: ");
scanf("%d", &n);
if (Rear == (MAX - 1))
{
printf("\nQueue Is Full\n");
}
else
{
if (Front == -1)
{
Front = 0;
}
Rear++;
q[Rear] = n;
printf("%d added to the queue\n", n);
}
```

```
}
void Dequeue()
{
if (Front == -1 && Rear == -1)
{
printf("\nQueue is Empty\n");
}
else
{
n = q[Front];
printf("\nElement %d is deleted\n", n);
Front++;
if (Front > Rear)
{
Front = Rear = -1;
}
}
}
void display()
{
int i;
if (Front == -1 && Rear == -1)
{
printf("Queue is Empty\n");
}
else
```

```
{
printf("\nQueue Elements: ");
for (i = Front; i <= Rear; i++)
{
printf("%d ", q[i]);
}
printf("\n");
}
}
int main()
{
Enqueue();
Enqueue();
Enqueue();
Enqueue();
display();
Dequeue();
display();
return 0;
}
```

Output:

```
Enter a Number: 50
50 added to the queue
Enter a Number: 51
51 added to the queue
Enter a Number: 52
52 added to the queue
Enter a Number: 53
53 added to the queue

Queue Elements: 50 51 52 53

Element 50 is deleted

Queue Elements: 51 52 53
```

Conclusion:

1.)What is the structure of queue ADT?

→ A queue is a linear data structure that follows the First-In-First-Out (FIFO) principle. It operates like a line where elements are added at one end (rear) and removed from the other end (front).

Basic Operations of Queue Data Structure

1. Enqueue (Insert): Adds an element to the rear of the queue.

2. Dequeue (Delete): Removes and returns the element from the front of the queue.

3. Peek: Returns the element at the front of the queue without removing it.

4. Empty: Checks if the queue is empty.

5. Full: Checks if the queue is full.

2.)List various applications of queues?

➔ Here are various applications of queues:

1. **CPU Scheduling**: In operating systems, queues are used in processes like round-robin scheduling, where processes are scheduled based on their arrival in a queue.
2. **Printer Spooling**: Print jobs are queued in a buffer and executed in order, ensuring efficient printing without conflicts.
3. **Call Centers**: Queues manage customer calls, routing them to available agents in the order they were received.
4. **Breadth-First Search (BFS)**: In graph traversal algorithms like BFS, a queue is used to explore nodes layer by layer.
5. **Traffic Management**: Queues are used in simulations of traffic systems, such as for toll booths or traffic signals.
6. **Data Stream Management**: Streaming data like videos or network packets are often managed using queues for buffering.

3.)Where is the queue used in a computer system processing?

➔ Queues are used in various aspects of computer system processing for managing tasks and data efficiently. Here are key areas where queues play a role:

1. CPU Task Scheduling: The operating system uses a queue to manage tasks that need CPU time. Processes are placed in a queue and scheduled based on algorithms like First-Come, First-Served (FCFS) or Round-Robin.
2. Disk Scheduling: Disk I/O requests are placed in a queue, and the operating system manages them to optimize disk access times (e.g., using algorithms like SCAN or LOOK).
3. Input/Output (I/O) Management: In systems dealing with high volumes of data input or output, queues buffer data streams between fast processors and slow I/O devices, allowing smoother data flow.
4. Print Queue: In printer spooling, print jobs are queued, ensuring that jobs are processed sequentially based on their arrival.
5. Network Data Packet Processing: Queues are used in network routers and switches to handle incoming and outgoing data packets, ensuring that packets are processed in the correct order.