Experiment No.4
Implementation of Queue menu driven program using arrays
Name: Shravani Sandeep Raut
Roll No: 48
Date of Performance:
Date of Submission:

Experiment No. 4: Simple Queue Operations

Aim: To implement a Linear Queue using arrays.

Objective:

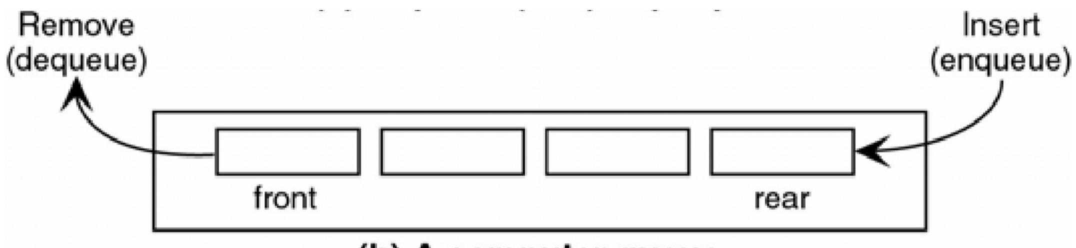
1 Understand the Queue data structure and its basic operations.

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:

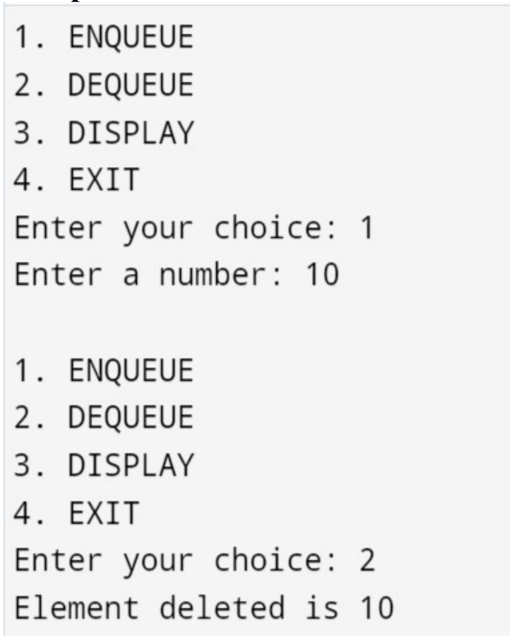
```
//Array implementation of  Linear Queue
#include <stdio.h>
#include <conio.h>
#define MAX 100
int q[MAX], n , i, front = -1, rear =-1, d;
void enqueue()
{
   if(rear == MAX -1)
   {
      printf("Queue is full");
   }
   else
   {
      printf("Enter a number: ");
      scanf("%d", &n);
      if(front == -1)
      {
         front++;
      }
      rear++;
      q[rear] = n;
   }
}

void dequeue()
{
   if(front == -1 && rear ==-1)
   {
      printf("Empty queue");
   }
   else
   {
      d = q[front];
      printf("Element deleted is %d",d);
      front++;
   }
}

void display()
{
   if(front == -1 && rear ==-1)
   {
      printf("Empty queue");
   }
   else
   {
        for(i = front; i<=rear; i++)
        {
            printf("%d\t",q[i]);
        }
    }
}
void main()
{
    int m=0;
    do{
    printf("\n1. ENQUEUE");
    printf("\n2. DEQUEUE");
    printf("\n3. DISPLAY");
    printf("\n4. EXIT");
    printf("\nEnter your choice: ");
    scanf("%d", &m);
    switch(m)
    {
            case 1: enqueue();
            break;
            case 2: dequeue();
            break;
            case 3: display();
            break;
    }

   }while(m != 4);
}
```

Output:



```
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 1
Enter a number: 10

1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 2
Element deleted is 10
```

Conclusion:

What is the structure of queue ADT?

- **Queue ADT** follows the **FIFO** (First In, First Out) principle.
- **Enqueue**: Adds an element at the rear.
- **Dequeue**: Removes an element from the front.
- **Peek/Front**: Retrieves front element without removal.
- **IsEmpty**: Checks if the queue is empty.
- **IsFull**: Checks if the queue is full.
- Can be implemented using arrays, linked lists, or circular queues.

List various applications of queues?

- **CPU Scheduling**: Queues are used to manage tasks waiting for CPU time in operating systems.
- **Print Spooling**: Documents waiting to be printed are queued in the order they are received.
- **Breadth-First Search (BFS)**: Used in graph traversal algorithms.
- **Handling Interrupts**: Queues are used to manage hardware and software interrupts.

Where is queue used in a computer system proceesing?

Queues are widely used in computer system processing in the following areas:

CPU Scheduling: Queues manage processes waiting for CPU time, ensuring fair and efficient execution of multiple tasks.

Job Scheduling: In batch processing systems, jobs are queued for execution based on priority and resource availability.

I/O Management: Queues hold requests for input/output operations, allowing the system to handle them sequentially and manage data flow efficiently.

Memory Management: Queues are used in page replacement algorithms to track pages in memory and manage virtual memory.