Experiment No.4
Implementation of Queue menu driven program using arrays
Name: Dnyanesh Baburao Panchal
Roll No: 34
Date of Performance:
Date of Submission:
Marks:
Sign:

Experiment No. 4: Simple Queue Operations

Aim: To implement a Linear Queue using arrays.

Objective:

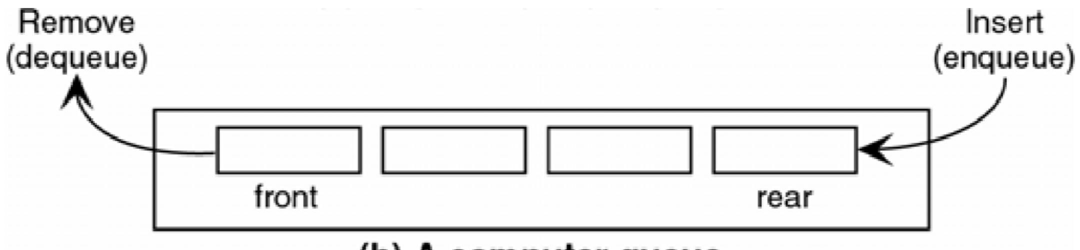
1 Understand the Queue data structure and its basic operations.

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
```

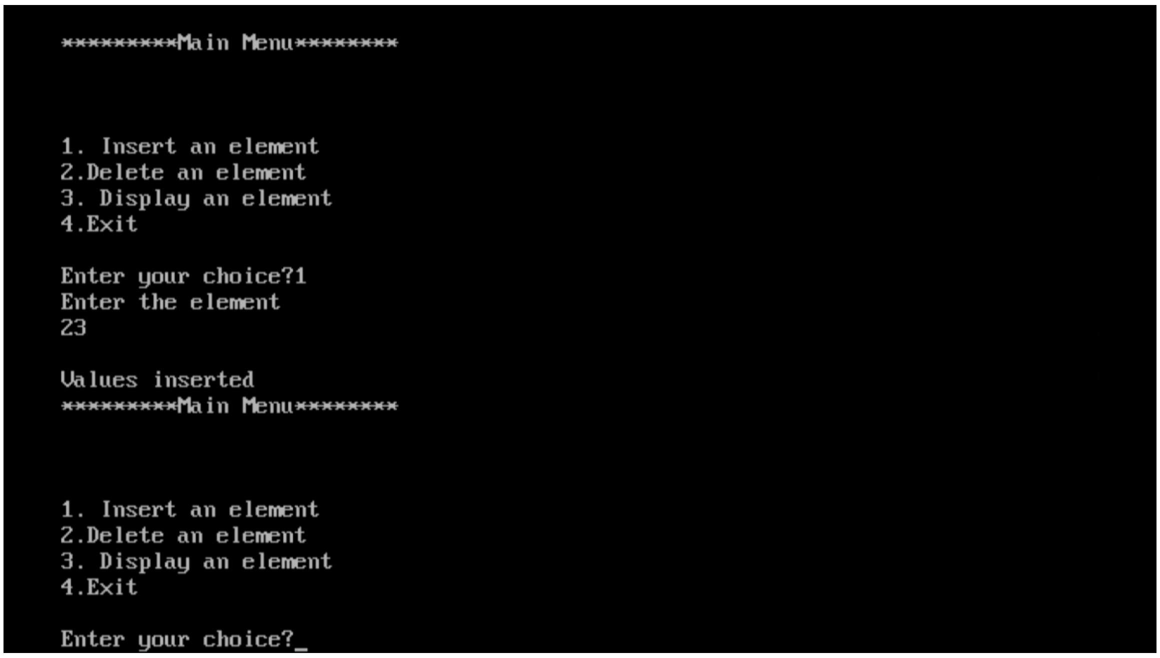
Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#define maxsize 5

void insert();
void deleted();
void display();
int front=-1,rear=-1;
int queue[maxsize];

void main()
{
        int choice;
        clrscr();
        while(choice!=4)
        {
                printf("\n*********Main Menu********\n");
                printf("\n                                             \n");
                printf("\n1. Insert an element\n2.Delete an element\n3. Display an
element\n4.Exit\n") ;
                printf("\nEnter your choice?");
                scanf("%d",&choice);

                switch(choice)

                {

                        case 1:

                        insert();

                        break;

                        case 2:

                        deleted();

                        break;

                        case 3:

                        display();

                        break;

                        case 4:

                        exit(0);

                        break;

                        default:

                        printf("\nEnter valid choice??\n");

                }

        }

        getch();

}

void insert()

{

        int item;

        printf("\Enter the element\n");

        scanf("\n%d",&item);

        if(rear==maxsize-1)
        {
                printf("\nOVERFLOW\n");
                return;
        }
        else if(front==-1 && rear==-1)
        {
                front=0;
                rear=0;
        }
        else
        {
                rear=rear+1;
        }
        queue[rear]=item;
        printf("\nValues inserted");
}

void deleted()
{
        int item;
        if(front==-1||front>rear)
        {
                printf("\nUNDERFLOW\n");
                return;
        }
        else
        {
```

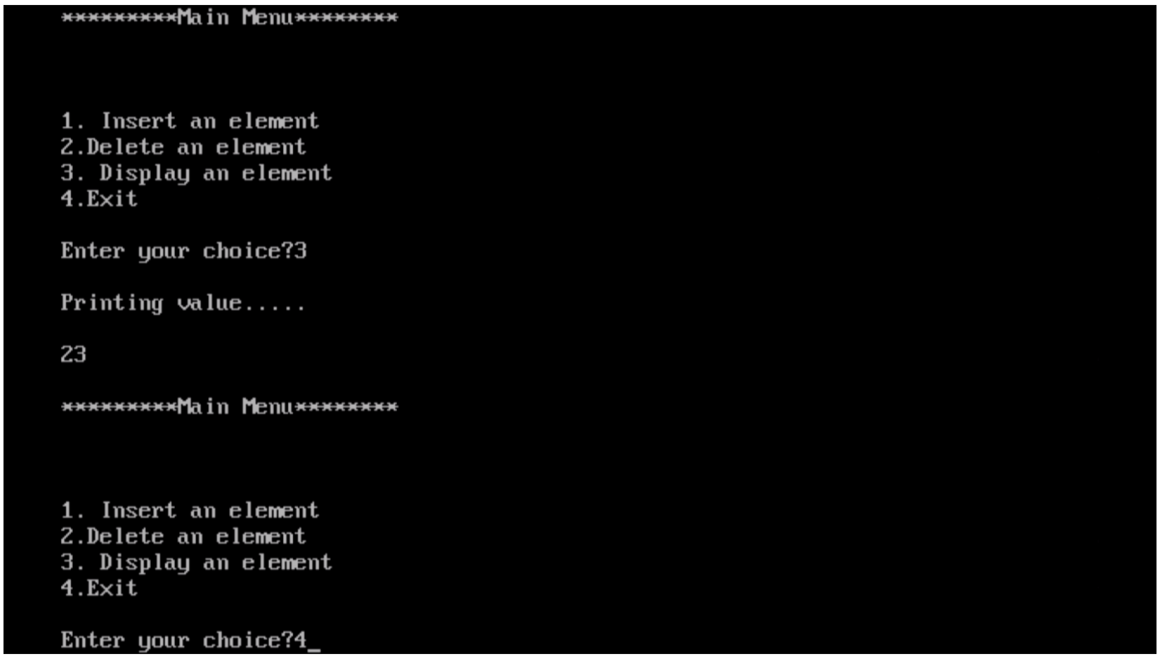
```
        item=queue[front];

        if(front==rear)
        {
            front=-1;
            rear=-1;
        }
        else
        {
            front=front+1;
        }
        printf("\n value deleted");
    }
}

void display()
{
    int i;
    if(rear==-1)
    {
        printf("\nEmpty queue\n");
    }
    else
    {
        printf("\nPrinting value.....\n");
        for(i=front;i<=rear;i++)
        {
```

```
                        printf("\n%d\n",queue[i]);
                }
        }
}
```

Output:



```
*********Main Menu*********



1. Insert an element
2.Delete an element
3. Display an element
4.Exit

Enter your choice?1
Enter the element
23

Values inserted
*********Main Menu*********



1. Insert an element
2.Delete an element
3. Display an element
4.Exit

Enter your choice?_
```


```
*********Main Menu********



1. Insert an element
2.Delete an element
3. Display an element
4.Exit

Enter your choice?3

Printing value.....

23

*********Main Menu********



1. Insert an element
2.Delete an element
3. Display an element
4.Exit

Enter your choice?4_
```

Conclusion:

1)What is the structure of queue ADT?

➢ The structure of a Queue ADT (Abstract Data Type) typically consists of a linear data structure in which elements are added to the back (rear) and removed from the front (front). It follows the First-In-First-Out (FIFO) principle, meaning that the element inserted first is the first one to be removed. In the provided C code, the queue is implemented as an array with operations like insertion, deletion, and display. It maintains a front and rear pointer to manage the queue.

2)List various applications of queues?

➢ Various applications of queues include:

- Task scheduling in operating systems, where processes are managed in a queue.

- Print job management, where print requests are processed in the order they are received.

- Breadth-first search in graph algorithms.

- Handling requests in web servers.

- Buffer management in data communication systems.

- Call center systems to manage customer service requests.

- Simulations and modeling in various scientific and engineering applications.

- Handling requests in shared resources like CPU scheduling.

3)Where is queue used in a computer system proceesing?

➢ Queues are used in various aspects of computer system processing:

-Process Scheduling: Operating systems use queues to manage the execution of processes. The ready queue holds processes waiting for CPU time.

- Print Queue: In printing systems, documents are added to a print queue and processed in the order they are received.

- Task Management: Task or job queues are used in multi-threaded or multi-core environments to manage the execution of tasks.

- Interrupt Handling: Queues are employed to manage interrupts, allowing the CPU to process them in a sequential manner.

- Buffering: In input/output systems, data is often buffered in queues before it is processed.

- Data Structures: Queues are used in various data structures, such as in breadth-first search, which is a fundamental graph traversal algorithm.

In summary, queues play a crucial role in managing tasks and data flow in computer systems and are applied in a wide range of applications to ensure orderly and efficient processing of tasks and requests.