Experiment No.4
Implementation of Queue menu driven program using arrays
Name: Swarup Satish Kakade
Roll No:19
Date of Performance:
Date of Submission:
Marks:
Sign:

Experiment No. 4: Simple Queue Operations

Aim: To implement a Linear Queue using arrays.

Objective:

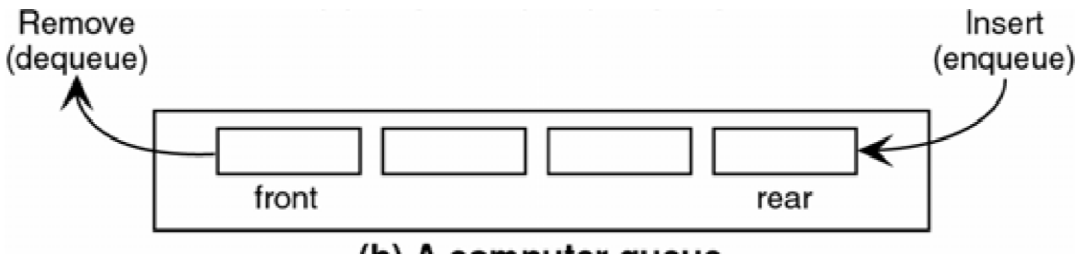
1 Understand the Queue data structure and its basic operations.

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];
```

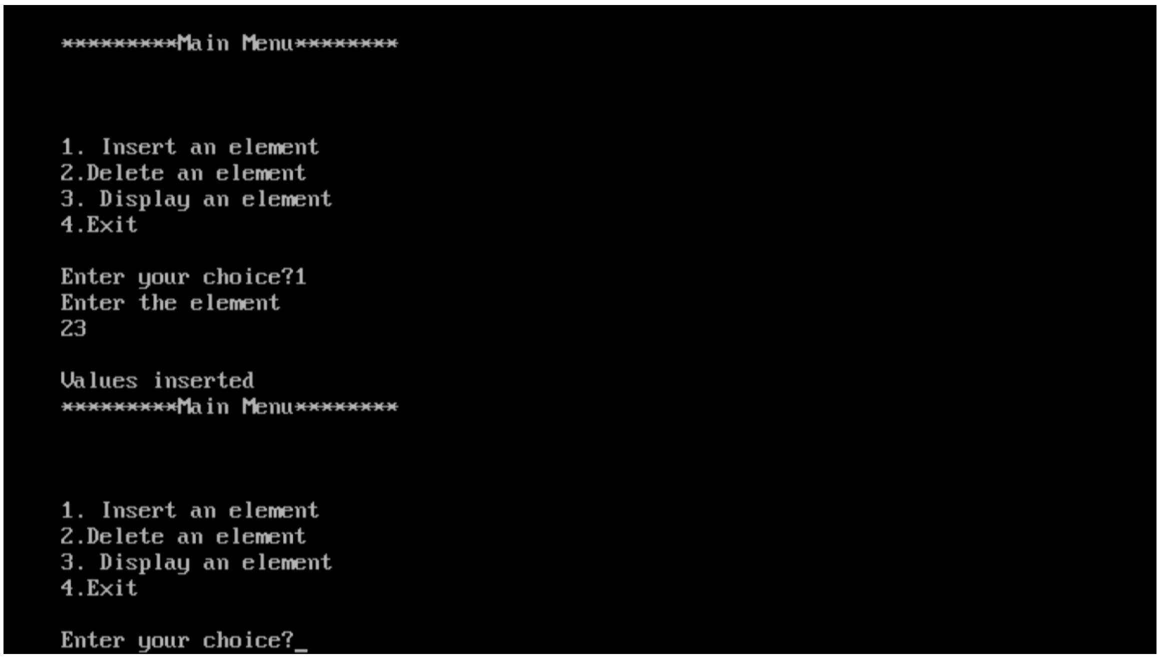
```
void main()
{
        int choice;
        clrscr();
        while(choice!=4)
        {
                printf("\n*********Main Menu********\n");
                printf("\n                                   \n");
                printf("\n1. Insert an element\n2.Delete an element\n3. Display an
element\n4.Exit\n") ;
                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
        {
```

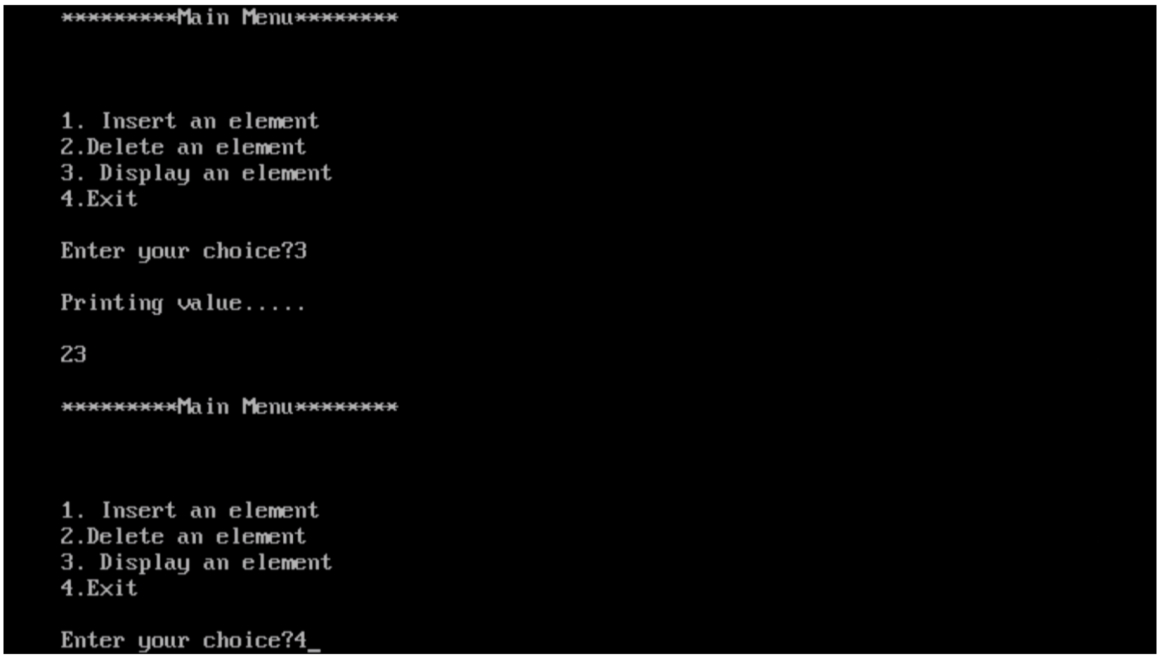
```
		rear=rear+1;
	}
	queue[rear]=item;
	printf("\nValues inserted");
}

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 Queue Abstract Data Type (ADT) is a linear data structure that follows the "First-In-First-Out" (FIFO) principle, meaning that the first element added to the queue is the first one to be removed. The basic structure of a Queue ADT typically includes the following operations and properties:

- **enqueue(item):** Adds an item to the back (rear) of the queue.
- **dequeue():** Removes and returns the item from the front of the queue.
- **front():** Returns (without removing) the item at the front of the queue.
- **isEmpty():** Checks if the queue is empty and returns a boolean value.
- **size():** Returns the number of elements currently in the queue.
- **clear():** Removes all elements from the queue, leaving it empty.

2) List various applications of queues?

- **Print Queue:** Managing print jobs in printers, ensuring that they are processed in the order they were requested.
- **Breadth-First Search (BFS):** Queue data structures are used in graph algorithms like BFS to explore nodes level by level.
- **Order Processing:** E-commerce websites use queues to manage orders, ensuring orders are processed in the order they are received.
- **Traffic Management:** Traffic lights and intersections use queues to manage the order of vehicles at stoplights.

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

- **Task Scheduling:** Operating systems use queues to manage processes and tasks in the CPU scheduler. Tasks in the ready state are placed in a queue, and the scheduler determines the order in which they are executed based on priority and other scheduling algorithms.
- **Memory Management:** Memory allocation and deallocation may involve queues, particularly in scenarios where memory blocks are allocated from and returned to a pool of available memory.
- **Caching:** Queues can be used to implement caching mechanisms, where items are evicted from the cache based on their position in the queue (e.g., LRU caching).
- **I/O Management:** Input and output operations in a computer system often involve queues. For example, a device driver may place I/O requests in a queue to ensure they are executed in the order received, optimizing the use of I/O devices.