Experiment No.4
Implementation of Queue menu driven program using arrays
Name: Nitish Jha
Roll No:18
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.

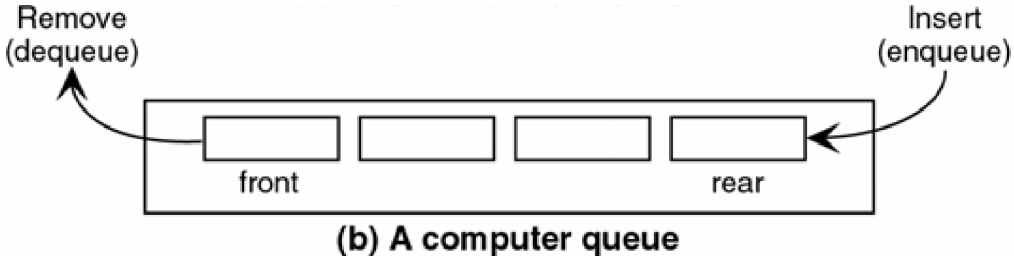
2. Understand the method of defining Queue ADT and its basic operations.

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;
```

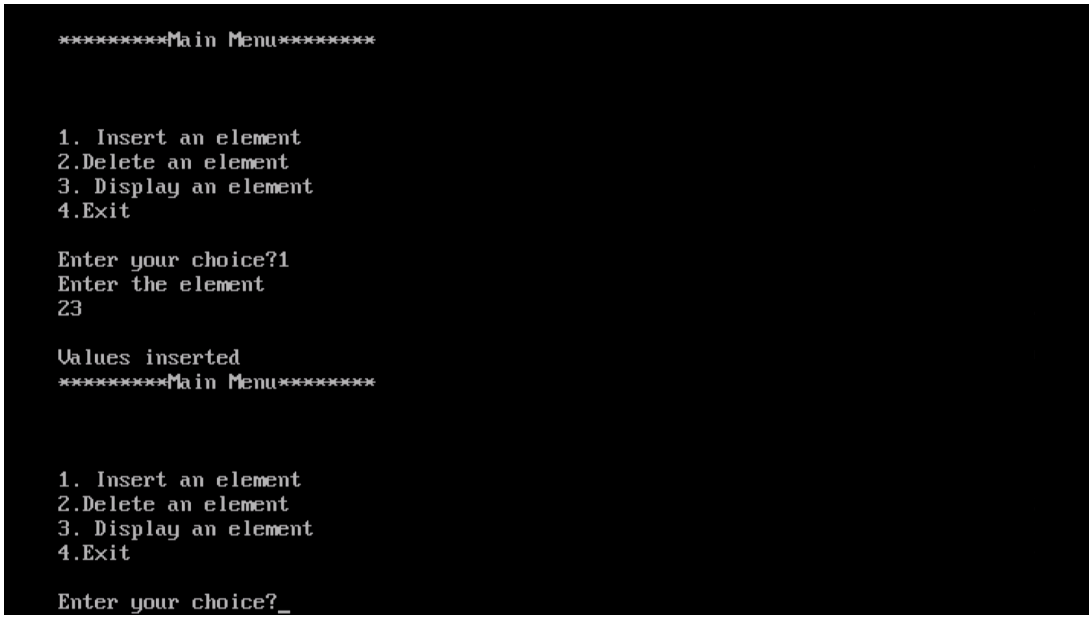
```
	clrscr();
	while(choice!=4)
	{
		printf("\n*********Main Menu********\n");
		printf("\n                                        \n");
		printf("\n1. Insert an element\n2.Delete an element\n3. Display an
element\n4.Exit\n") ;
		printf("\nEnter your choice?");
		scanf("%d",&choice);
		switch(choice)
		{
			case 1:
			insert();
			break;
			case 2:
			deleted();
			break;
			case 3:
			display();
			break;
			case 4:
			exit(0);
			break;
			default:
			printf("\nEnter valid choice??\n");
		}
	}
	getch();
```

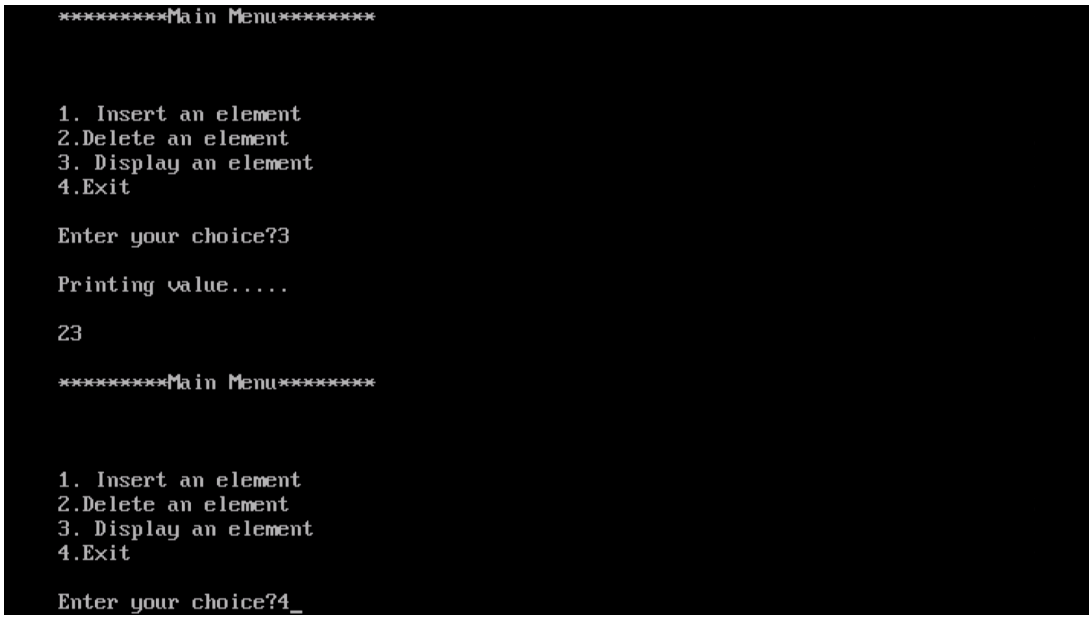
```
}
void insert()
{
	int item;
	printf("\Enter the element\n");
	scanf("\n%d",&item);
	if(rear==maxsize-1)
	{
		printf("\nOVERFLOW\n");
		return;
	}
	else if(front==-1 && rear==-1)
	{
		front=0;
		rear=0;
	}
	else
	{
		rear=rear+1;
	}
	queue[rear]=item;
	printf("\nValues inserted");
}

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:





Conclusion:

1)What is the structure of queue ADT?

- A Queue Abstract Data Type (ADT) is fundamentally organized as a linear structure where elements find their way into the back (or rear) and exit from the front. This orderly arrangement adheres to the First-In-First-Out (FIFO) principle, signifying that the element initially inserted is the first to exit. In the given C code, the queue is embodied as an array, equipped with functions for insertion, deletion, and display. This queue is efficiently managed through the orchestration of front and rear pointers.

2)List various applications of queues?

- Queues serve a multitude of practical purposes in various domains, encompassing:

- Operating systems employ queues for task scheduling, effectively managing the order in which processes are executed.

- Print job management systems process print requests systematically, adhering to the received order.

- Graph algorithms, particularly breadth-first search, utilize queues to explore nodes.

- Web servers handle incoming requests in an orderly manner, thanks to queues.

- Data communication systems rely on queues for efficient buffer management.

- Call center systems streamline customer service requests using queues.

- Scientific and engineering applications employ queues for simulations and modeling.

- Shared resources, such as CPU scheduling, efficiently manage incoming requests with queues.

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

- Queues find versatile utility within computer systems across various domains:

- Process Scheduling: Operating systems skillfully employ queues for process execution management. The ready queue serves as a repository for processes awaiting their turn for CPU execution.

- Print Queue: In the realm of printing systems, documents are systematically enqueued and processed on a first-come, first-served basis.

- Task Management: In multi-threaded or multi-core environments, task or job queues adeptly oversee task execution.

- Interrupt Handling: Queues prove indispensable for handling interrupts, allowing the CPU to address them sequentially.

- Buffering: Input/output systems make efficient use of queues to buffer data prior to processing.

- Data Structures: Queues are a foundational element within various data structures, prominently featured in fundamental graph traversal algorithms like breadth-first search.

To summarize, queues emerge as pivotal components in task and data flow management within computer systems, spanning a multitude of applications to ensure structured and efficient task and request processing.