Experiment No.4
Implementation of Queue menu driven program using arrays
Name: Jaffari Mohammed Ali Sayyed Naqi Ali
Roll No:16
Date of Performance:
Date of Submission:
Marks:
Sign:

Experiment No. 4: Simple Queue Operations

Aim: To implement a Linear Queue using arrays.

Objective:

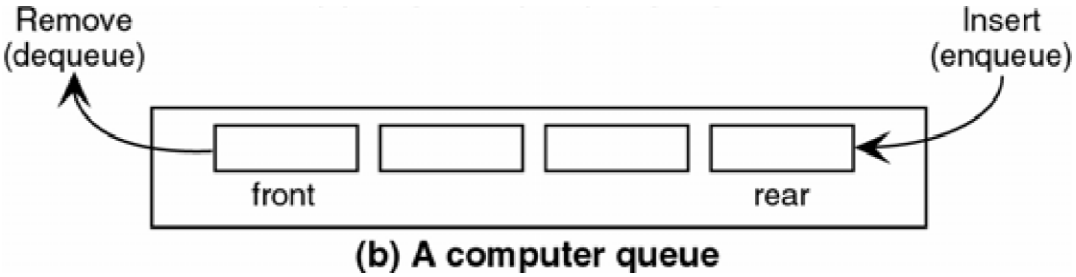
1 Understand the Queue data structure and its basic operations.

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 item that is added first will be the first one to be removed. The basic structure of a Queue ADT typically includes two main operations:

1. Enqueue (Add): This operation is used to add (enqueue) an element to the back of the queue.

2. Dequeue (Remove): This operation is used to remove (dequeue) the element from the front of the queue.

In addition to these fundamental operations, a Queue ADT may also include the following methods and attributes:

- Front (or Peek): This operation allows you to view the element at the front of the queue without removing it.
-
- - IsEmpty:This method checks whether the queue is empty or not.
-
- - Size (or Length):*This method returns the number of elements in the queue.

A Queue ADT can be implemented using various data structures, including arrays and linked lists. The choice of implementation can affect the efficiency of enqueue and dequeue operations. Queues are widely used in various computer science applications, such as scheduling processes, managing tasks, and handling data in a first-in, first-out manner.

2)List various applications of queues?

- Queues find applications in various domains, such as:

1. Task Management: They are used for scheduling and managing tasks in operating systems, call centers, and background job processing.
2. Data Flow: Queues buffer data in applications like print spooling, networking, and background task processing.

3. Event Handling: Event-driven systems use queues to manage and process events.
4. Resource Allocation: They ensure fair resource allocation in scenarios like printer queues and job scheduling.
5. Traffic Control: Queues are instrumental in managing traffic flow and order in traffic lights and intersections.
6. Order Processing: In retail and e-commerce, orders are queued for efficient fulfillment.

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

In a computer system, queues serve as crucial tools for task management, resource allocation, and efficient data flow. They are utilized in various contexts, such as:

- Task Scheduling: Queues are central to an operating system's task scheduler, ensuring processes are executed fairly and efficiently.
- Print Management: Print queues are employed to orderly handle print jobs, allowing for a first-come, first-served printing approach.
- Background Processing: Queues facilitate the execution of background tasks, such as data processing and maintenance, without disrupting foreground operations.
- Multithreading: Queues are instrumental in distributing tasks among threads, enabling parallel processing and efficient multitasking.
- Job Scheduling: Job queues determine the execution order of tasks, often used in batch processing and job submission systems.
- Request Handling: In web and application servers, queues manage incoming requests, ensuring orderly and efficient processing.
- Network Data Management: Queues help buffer data packets in network devices, preventing data congestion and ensuring smooth data transmission.
- Event Handling: Event queues are used in event-driven applications to manage user interactions and system notifications.
- Printers and Fax: Printers and fax machines use queues to store and process print jobs, optimizing document management.