Experiment No.4
Implementation of Queue menu driven program using arrays
Name:Ayush Gupta
Roll No:14
Date of Performance:
Date of Submission:
Marks:
Sign:

Experiment No. 4: Simple Queue Operations

Aim: To implement a Linear Queue using arrays.

Objective:

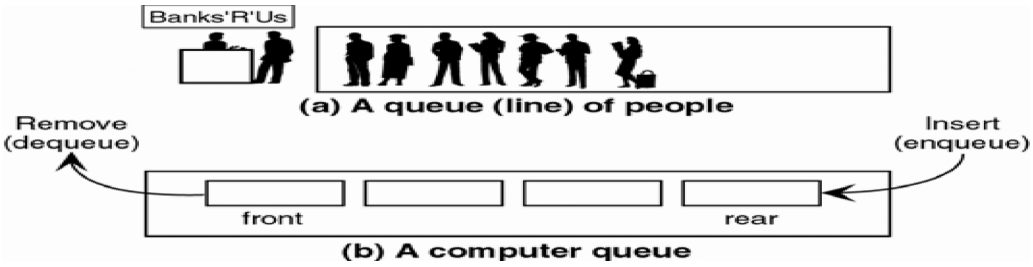
1 Understand the Queue data structure and its basic operations.

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.



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:

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. In a queue, elements are added at the rear (enqueue) and removed from the front (dequeue). The basic operations associated with a queue are:

1. Enqueue (or Insert): Add an element to the rear of the queue.

2. Dequeue (or Delete): Remove and return the element from the front of the queue.

3. Front (or Peek): View the element at the front of the queue without removing it.

4. IsEmpty: Check if the queue is empty.

5. Size (or Length): Return the number of elements currently in the queue.

The structure of a queue can be implemented using various underlying data structures. The two most common methods are:

1. Array-based Queue

- In this implementation, a queue is created using an array. Elements are added at the rear and removed from the front. When elements are dequeued, the remaining elements are shifted to maintain the order. This can lead to inefficiencies in terms of time complexity, especially for large queues.

2. Linked List-based Queue

- In this implementation, a queue is created using a linked list data structure. Each element in the queue is represented by a node, and these nodes are connected in a way that maintains the order of the queue. Elements are enqueued at the rear by adding a new node and dequeued from the front by removing the front node. This implementation doesn't have a fixed size and can dynamically grow or shrink as needed.

Here's a basic representation of the structure of a queue in pseudo-code:

Queue ADT:

- Data: A linked list containing nodes.

- Operations:

- Enqueue(element): Add an element to the rear of the queue.

- Dequeue(): Remove and return the element from the front of the queue.

- Front(): Return the element at the front of the queue without removing it.

- IsEmpty(): Check if the queue is empty.

- Size(): Return the number of elements in the queue.

In practice, you would implement the Queue ADT in a programming language of your choice using the chosen data structure (array or linked list) and its associated operations. Queues are widely used in various applications, including task scheduling, data buffering, and process management.

List various applications of queues?

Here are various concise applications of queues:

1. Task scheduling in operating systems.

2. Print job management and spooling.

3. Breadth-First Search (BFS) in graph algorithms.

4. Data buffering and processing.

5. Task management in multitasking environments.

6. Request handling in web servers.

7. Order processing and e-commerce.

8. Background task execution.

9. Interprocess communication and synchronization.

10. Task priority management.

11. Call center systems and customer support.

12. Resource management and access control.

13. Web caching and content retrieval.

14. Network routing in networking.

15. Real-time task scheduling in embedded systems.

Where is queue used in a computer system proceesing?

Queues are used in computer systems for:

1. Task scheduling and multitasking.
2. I/O request handling and print spooling.
3. Background task execution.
4. Interprocess communication.
5. Request handling in web servers.
6. Job and print job management.
7. Resource access control.
8. Real-time systems and embedded systems.
9. Network routing and data transmission.
10. File I/O buffering.
11. Email filtering and more.