Experiment No.4
Implementation of Queue menu driven program using arrays
Name:Rohan Ganpat Mangaonkar
Roll No:27
Date of Performance:8/8/23
Date of Submission:22/8/23
Marks:
Sign:

Experiment No. 4: Simple Queue Operations

Aim: To implement a Linear Queue using arrays.

Objective:

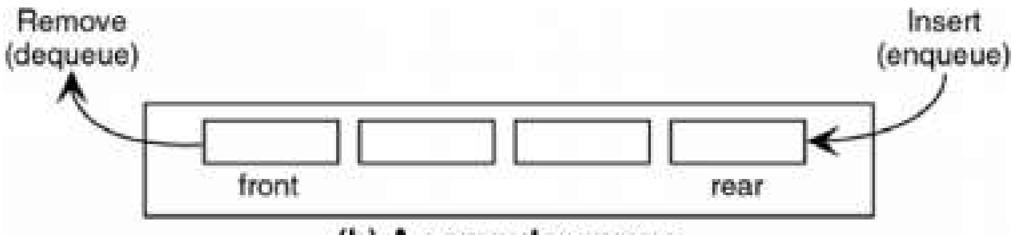
1 Understand the Queue data structure and its basic operations.

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:

```
 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?4
```

Conclusion:

Q1 What is the structure of queue ADT?

A Queue Abstract Data Type (ADT) is typically structured as follows:

1. Enqueue: This operation is used to add an element to the back or rear of the queue. It is also called "push" in some contexts.

2. Dequeue: This operation removes and returns the element at the front of the queue. It is also called "pop" in some contexts.

3. Front: This operation retrieves the element at the front of the queue without removing it.

4. IsEmpty: This operation checks if the queue is empty. It returns a boolean value (true if the queue is empty, false if it's not).

5. Size: This operation returns the number of elements currently in the queue.

Queues follow a "first-in, first-out" (FIFO) order, which means that the first element added to the queue will be the first to be removed (via dequeue), and the last element added will be the last to be removed.

Queues can be implemented using various data structures like arrays or linked lists. The choice of implementation depends on the specific use case and the required operations' time complexity.

Q2 List various applications of queues?

Task Scheduling

Breadth First Searching

Customer Service

Print Job Management

Request Handling

Q3 Where is queue used in a computer system proceesing?

Queues are commonly used in computer systems and processing for various purposes, including:

1. Task Scheduling: Operating systems use queues to manage processes and their execution order. For example, a ready queue holds processes waiting to be executed by the CPU.

2. Print Spooling: Print jobs are often placed in a queue (print queue) to be processed in the order they were received.

3. Disk I/O: Requests for disk read/write operations are often managed in queues to ensure efficient access and prevent data conflicts.

4. Network Packets: In networking, packets are often placed in queues for transmission, ensuring data is sent and received in an orderly manner.

5. Message Queues: In distributed systems and interprocess communication, message queues facilitate the exchange of data between different parts of a system.

6. Task Management: Task queues are used for managing asynchronous tasks in applications, ensuring they are processed in a controlled manner.

These are just a few examples of how queues are used in computer systems to manage and control various processes and data flows. They help in organizing and prioritizing tasks, ensuring efficient resource utilization, and maintaining the integrity of data flows