Experiment No.4
Implementation of Queue menu driven program using arrays
Name: Gautam D. Chaudhari
Roll No: 04
Date of Performance:
Date of Submission:
Marks:
Sign:

Experiment No. 4: Simple Queue Operations

Aim: To implement a Linear Queue using arrays.

Objective:

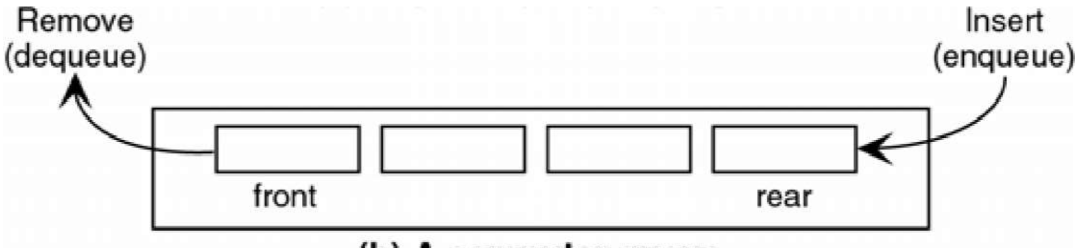
1 Understand the Queue data structure and its basic operations.

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
```

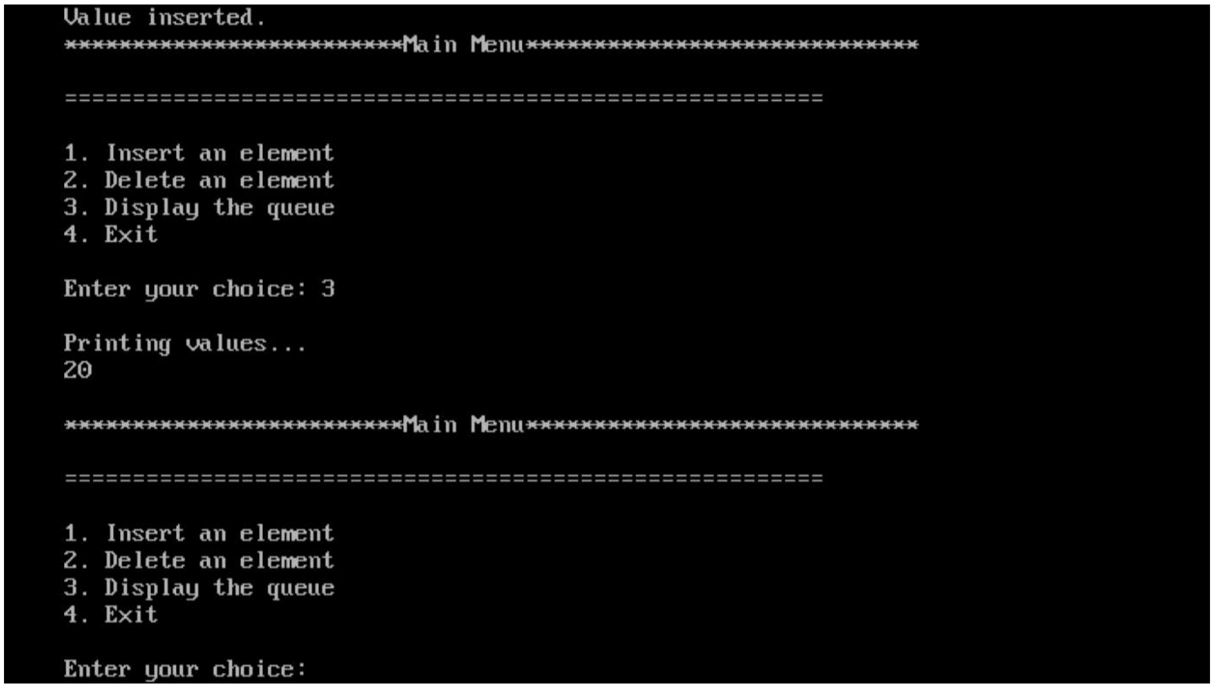
ISFULL()

```
1. If(rear = max)then
        return 1
2. return 0
```

Code:

```
#include <stdio.h>

#include <stdlib.h>


#define maxsize 5


void insert();

void delete();

void display();


int front = -1, rear = -1;

int queue[maxsize];


int main() {

    int choice;


    while (choice != 4) {

        printf("\n**************************Main
Menu*******************************\n");


printf("\n==============================================================\n");

        printf("\n1. Insert an element\n2. Delete an element\n3. Display the queue\n4. Exit\n");

        printf("\nEnter your choice: ");

        scanf("%d", &choice);
        switch (choice) {
          case 1:
            insert();
            break;
          case 2:
            delete();
            break;
          case 3:
            display();
            break;
          case 4:
            exit(0);
            break;
          default:
            printf("\nEnter a valid choice.\n");
        }
    }

    return 0;
}

void insert() {
    int item;
    printf("\nEnter the element: ");
    scanf("\n%d", &item);
    if (rear == maxsize - 1) {
        printf("\nQueue is full (OVERFLOW)\n");
        return;
    }

    if (front == -1 && rear == -1) {
        front = 0;
        rear = 0;
    } else {
        rear = rear + 1;
    }

    queue[rear] = item;
    printf("\nValue inserted.");
}

void delete() {
    int item;

    if (front == -1 || front > rear) {
        printf("\nQueue is empty (UNDERFLOW)\n");
        return;
    } else {
        item = queue[front];
        if (front == rear) {
            front = -1;
            rear = -1;
        } else {
            front = front + 1;
        }

        printf("\nValue deleted: %d\n", item);
    }
}

void display() {
    int i;

    if (rear == -1) {
        printf("\nEmpty queue\n");
    } else {
        printf("\nPrinting values...\n");
        for (i = front; i <= rear; i++) {
            printf("%d\n", queue[i]);
        }
    }
}
```

Output:



```
Value inserted.
*************************Main Menu*****************************

========================================================

1. Insert an element
2. Delete an element
3. Display the queue
4. Exit

Enter your choice: 3

Printing values...
20

*************************Main Menu*****************************

========================================================

1. Insert an element
2. Delete an element
3. Display the queue
4. Exit

Enter your choice:
```

Conclusion:

What is the structure of queue ADT?

A Queue is a crucial data structure in computer science, adhering to the First-In-First-Out (FIFO) principle. Its Abstract Data Type (ADT) comprises five main components:

- Enqueue: This operation appends an element to the back of the queue, often called "push" or "insert."
- Dequeue: Removing and returning the front element, typically referred to as "pop."
- Front: It allows peeking at the front element without removal.
- IsEmpty: Checking if the queue is empty is vital to prevent errors during dequeuing from an empty queue.
- Size: This method reports the count of elements in the queue.

The visual representation of a Queue ADT shows elements arranged in a line, with the front element being the first to leave the queue, and the rear element the last.

List various applications of queues?

Some common applications of queues include:

Task Scheduling: Operating systems use queues to manage tasks, ensuring that processes run in a fair and organized manner.

Print Job Management: Queues are used to prioritize and manage print jobs in printers, ensuring they are processed in the order they are received.

Breadth-First Search (BFS): Queues are essential in graph algorithms like BFS, where they help explore all neighbors of a node before moving on to their neighbors.

Web Servers: Queues can manage incoming web requests, ensuring that they are processed in the order they are received, preventing server overload.

Call Center Systems: Queues are used to manage incoming customer calls, directing them to available agents in the order they are received.

Buffering: In data communication, queues can be used to buffer data between two processes, ensuring a smooth flow of data.

Where is queue used in a computer system proceesing?

Queues are used in various aspects of a computer system for orderly processing. In operating systems, they are used for task scheduling, managing print jobs, and controlling resource access. Web servers use queues to handle incoming requests in the order received, ensuring fairness and preventing overloads. In multithreaded applications, queues coordinate tasks among threads. They are also crucial in managing background tasks in mobile and web applications. Additionally, message queues facilitate communication between distributed system components, and print queues ensure documents are printed in the order they were sent to the printer. Queues play a fundamental role in maintaining order and efficiency in computer systems.