NAME: GAURAV KISHOR PATIL

DIV: 2 ROLL NO:54

BATCH: C

Experiment No.4
Implement Linear Queue ADT using Array
Date of Performance:
Date of Submission:

Experiment No. 4: Impalement of Linear Queue ADT using Array

Aim: To implement a Queue using arrays.

Objective:

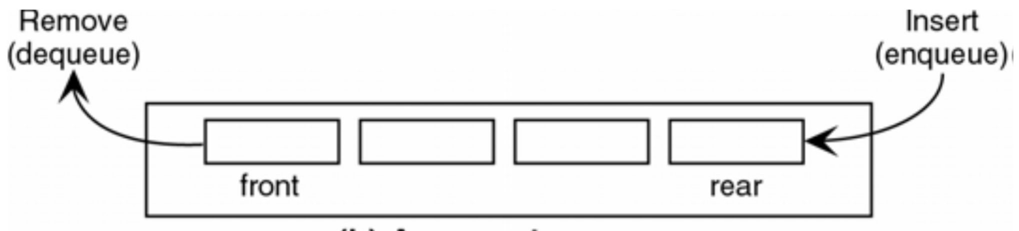
1 Understand the Queue data structure and its basis operations.

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

3. Learn how to create objects from an ADT and member function are invoked.

Theory:

A Queue is an ordered collection of items from which items may be deleted at one end (called the *front* of the queue) and into which items may be inserted at the other end (the *rear* of the queue). Queues remember things in first-in-first-out (FIFO) order. The basic operations in a queue are: Enqueue - Adds an item to the end of queue. Dequeue - Removes an item from the front



(b) A computer queue

A queue is implemented using a one dimensional array. FRONT is an integer value, which contains the array index of the front element of the array. REAR is an integer value, which contains the array index of the rear element of the array. When an element is deleted from the queue, the value of front is increased by one. When an element is inserted into the queue, the value of rear is increased by one.

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
```

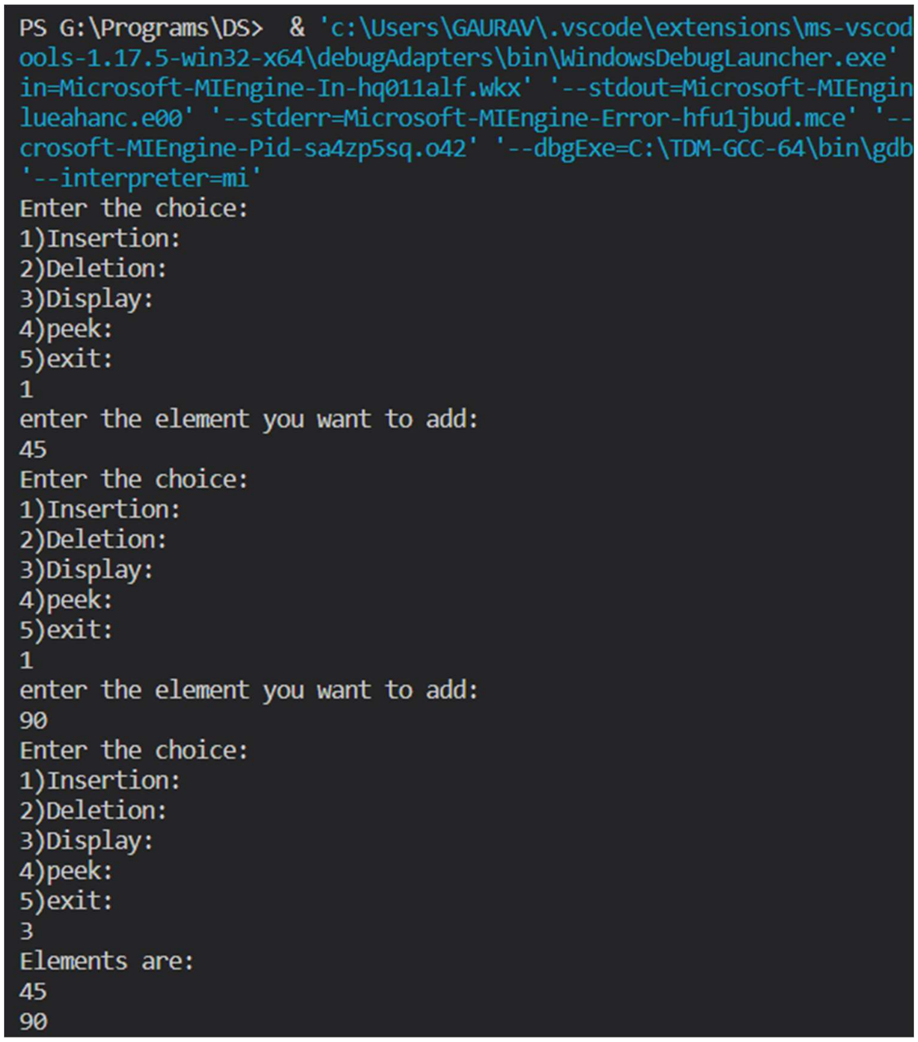
Code :

```
#include <stdio.h>
#include <string.h>

#define max 10

int queue[max];
int rear = -1, front = -1;

void enqueue(int);
void dequeue();
void display();
void peek();

void enqueue(int m) {
   if (rear >= max - 1) {
      printf("overflow");
   } else if (rear == -1 && front == -1) {
      front = rear = 0;
      queue[rear] = m;
   } else {
      rear++;
      queue[rear] = m;
```

```
    }
}

void dequeue() {
    if (front == -1 && rear == -1) {
        printf("underflow");
    } else if (front == rear) {
        printf("element popped is %d", queue[front]);
        front = rear = -1;
    } else {
        printf("element popped is %d", queue[front]);
        front++;
    }
}

void display() {
    int i = 0;
    if (front == -1 && rear == -1) {
        printf("Queue is empty\n");
    } else {
        printf("Elements are:\n");
        for (i = front; i <= rear; i++) {
            printf("%d\n", queue[i]);
        }
    }
}
```

```
void peek() {
    if (front == -1 && rear == -1) {
        printf("Queue is empty\n");
    } else {
        printf("The top element is %d\n", queue[front]);
    }
}

int main() {
    int x, e;

    do {
        printf("Enter the choice:\n");
        printf("1)Insertion:\n");
        printf("2)Deletion:\n");
        printf("3)Display:\n");
        printf("4)peek:\n");
        printf("5)exit:\n");

        scanf("%d", &x);
        switch (x) {
            case 1:
                printf("enter the element you want to add:\n");
                scanf("%d", &e);
                enqueue(e);
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
            case 4:
                peek();
                break;
            case 5:
                printf("you have exited the program\n");
                break;
            default:
                printf("wrong\n");
                break;
        }
    } while (x != 5);

    return 0;
}
```

Output



```
PS G:\Programs\DS>  & 'c:\Users\GAURAV\.vscode\extensions\ms-vscod
ools-1.17.5-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe'
in=Microsoft-MIEngine-In-hq011alf.wkx' '--stdout=Microsoft-MIEngin
lueahanc.e00' '--stderr=Microsoft-MIEngine-Error-hfu1jbud.mce' '--
crosoft-MIEngine-Pid-sa4zp5sq.o42' '--dbgExe=C:\TDM-GCC-64\bin\gdb
'--interpreter=mi'
Enter the choice:
1)Insertion:
2)Deletion:
3)Display:
4)peek:
5)exit:
1
enter the element you want to add:
45
Enter the choice:
1)Insertion:
2)Deletion:
3)Display:
4)peek:
5)exit:
1
enter the element you want to add:
90
Enter the choice:
1)Insertion:
2)Deletion:
3)Display:
4)peek:
5)exit:
3
Elements are:
45
90
```

Conclusion:

A queue in C is a versatile and data structure that overcomes the problems of insertion and deletion of elements whether from the front end or rear end. It follows the order of First In First Out (FIFO) and has the capability of determining the operations in a way that they can be enqueued and dequeued in any way. Queue data structure has a wide range of applications in computer science. Some of the common applications of Queue data structure are:

Task Scheduling: Queues can be used to schedule tasks based on priority or the order in which they were received.

Resource Allocation: Queues can be used to manage and allocate resources, such as printers or CPU processing time.

Batch Processing: Queues can be used to handle batch processing jobs, such as data analysis or image rendering.