Data Structures : Algorithms and Applications(Lab)

EXPERIMENT No:3

AIM: Implementation of Linear Queue Data Structure using array.

THEORY:

Queue Data Structure**:** A queue is a linear data structure that follows the First In, First Out (FIFO) principle. This means that the first element added to the queue will be the first one to be removed. It operates similarly to a real-world queue, such as a line of people waiting for service.

Operations:

1. Insertion (Enqueue): Adding an element to the rear of the queue.
2. Deletion (Dequeue): Removing an element from the front of the queue.
3. Display: Printing all elements in the queue from front to rear.

Implementation Using Arrays: In this implementation, a fixed-size array is used to represent the queue. The `front` and `rear` indices manage the current positions of the queue:

- `front`: Indicates the position from which elements are removed.
- `rear`: Indicates the position where elements are added.

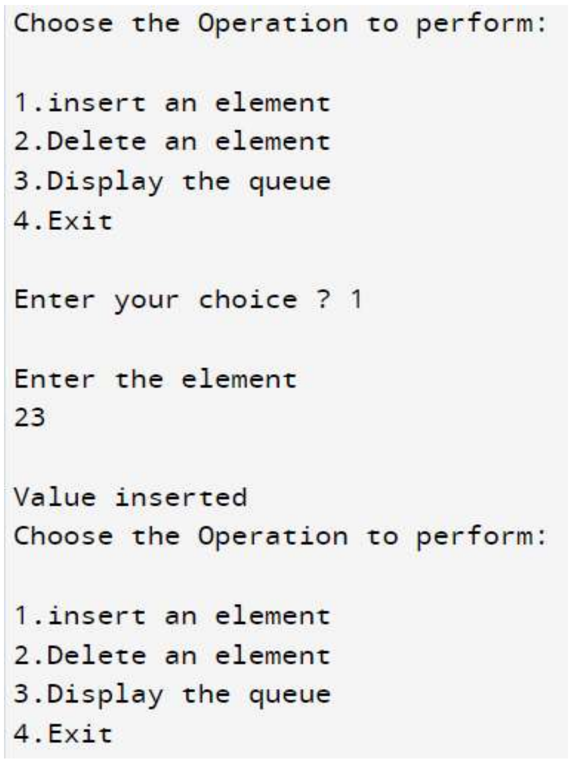
Overflow and Underflow:

- Overflow: Occurs when trying to insert an element into a full queue.
- Underflow**:** Occurs when trying to remove an element from an empty queue.

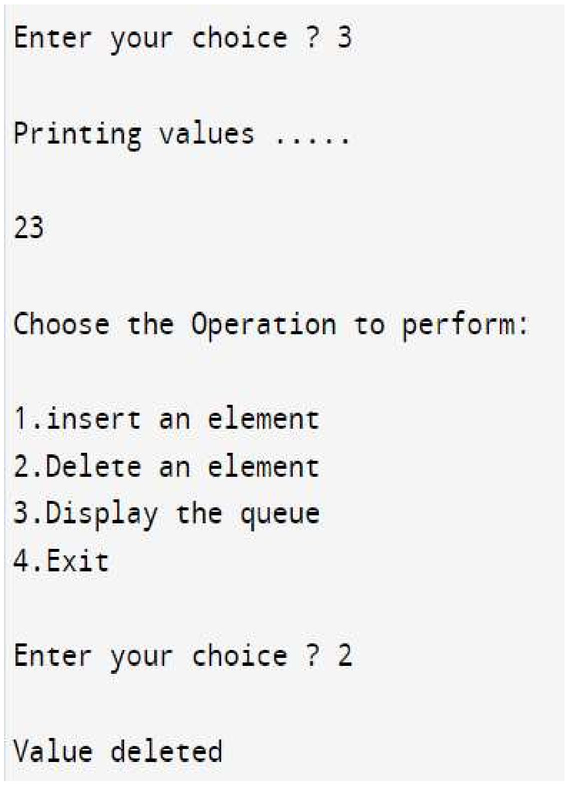
CODE:

```
#include <stdio.h>
#include <stdlib.h>
#define maxsize 5

void insert();
void delete();
void display();

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

void main() {
    int choice = 0;
    while (choice != 4) {
        printf("\nChoose the Operation to perform:\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 valid choice??\n");
        }
    }
}

void insert() {
    int item;
    printf("\nEnter the element\n");
    scanf("%d", &item);
    if (rear == maxsize - 1) {
        printf("\nOVERFLOW\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("\nUNDERFLOW\n");
        return;
    } else {
        item = queue[front];
        if (front == rear) {
            front = -1;
            rear = -1;
        } else {
            front = front + 1;
        }
        printf("\nValue deleted ");
    }
}

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

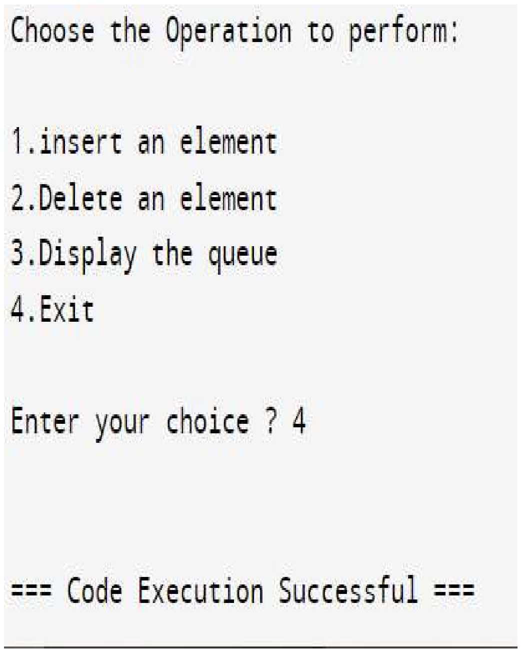
OUTPUT :



```
Choose the Operation to perform:

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

Enter your choice ? 1

Enter the element
23

Value inserted
Choose the Operation to perform:

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



```
Enter your choice ? 3

Printing values .....

23

Choose the Operation to perform:

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

Enter your choice ? 2

Value deleted
```



```
Choose the Operation to perform:

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

Enter your choice ? 4



=== Code Execution Successful ===
```

CONCLUSION:

An array-based queue manages data in a FIFO order, supporting operations like adding, removing, and displaying items. It's efficient for tasks requiring orderly processing and helps handle overflow and underflow conditions.