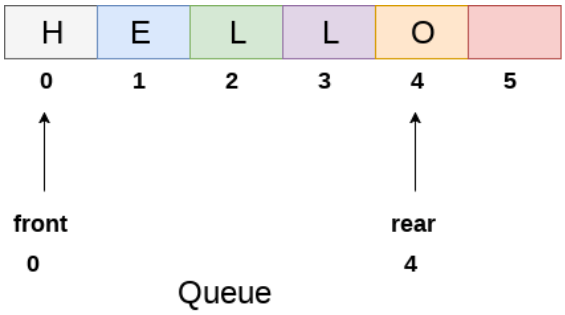
1. Introduction of queue what is Queues,
2. Linked Representation of Queue,
3. **Insertion & Deletion on Queue. ,**
4. **D- queue**,
5. **Priority Queue.**

---

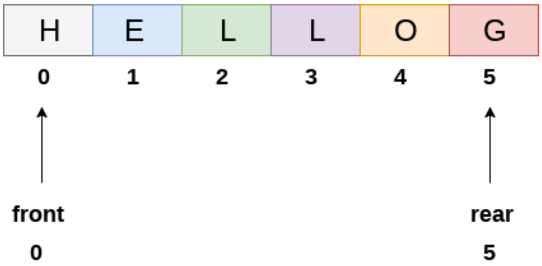
1. Introduction of queue what is Queues:

Queue is linear data structure where first element inserted will be process first hence it is also called as first in first out (FIFO) data structure. Queue representation using linear arrays is straightforward. For every queue, two variables, front and rear are utilized. These variables indicate the positions where insertions and deletions occur within the queue. Initially, both front and rear are set to -1, signifying an empty queue. The array representation of a queue with five elements, along with the corresponding values of front and rear, is illustrated in the figure below.



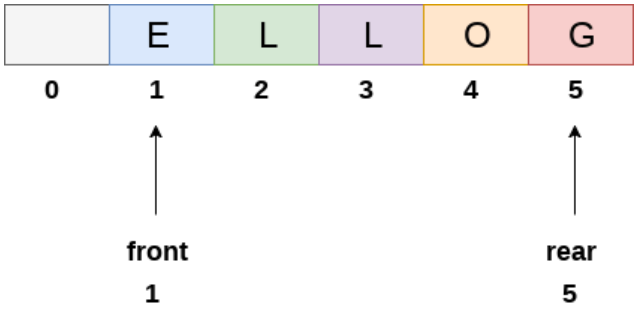
Queue

The above figure shows the queue of characters forming the English word **"HELLO"**. Since, No deletion is performed in the queue till now, therefore the value of front remains -1. However, the value of rear increases by one every time an insertion is performed in the queue. After inserting an element into the queue shown in the above figure, the queue will look something like following. The value of rear will become 5 while the value of front remains same.



Queue after inserting an element

After deleting an element, the value of front will increase from -1 to 0. However, the queue will look something like following.



Queue after deleting an element

Algorithm to delete item from queue:

Step 1: Check if the rear of the queue is at the maximum limit. If it is, show a message that says "OVERFLOW" and stop.

Step 2: If the front and rear are both at -1 (which means the queue is empty), then make both of them 0. Otherwise, move the rear one position ahead.

Step 3: Put the new number (NUM) in the queue at the rear.

Step 4: You're done! Exit the process.

Algorithm to Insert an Element in a Queue:

1. **Check if the Queue is Full:**
    - Before inserting an element, check if the queue is already full. You can do this by comparing the **rear** pointer to the maximum capacity (**maxsize - 1**). If **rear** is

equal to the maximum capacity, the queue is full and you cannot insert more elements.

2. **If Queue is Full, Display an Overflow Message:**
    - If the queue is full, display an "OVERFLOW" message and terminate the insertion process. Optionally, you can handle this condition as needed.
3. **If Queue is Not Full, Proceed with Insertion:**
    - If the queue is not full, proceed with the insertion:
        - Prompt the user to enter the element they want to insert.
        - Check if the queue is initially empty (both **front** and **rear** are -1).
        - If the queue is empty, set both **front** and **rear** to 0 to mark the first element in the queue.
        - If the queue is not empty, increment the **rear** pointer by 1 to make room for the new element.
        - Store the entered element in the queue at the updated **rear** position.
4. **Display a Confirmation Message:**
    - After successfully inserting the element, display a message confirming that the value has been inserted.

Menu driven program to implement queue using array

```
#include <iostream.h>

#include <conio.h>

#define maxsize 5


void insert();
```

```
void deleteItem();

void display();


int front = -1, rear = -1;

int queue[maxsize];


void main() {

    int choice;

    while (choice != 4) {

        clrscr();

cout<<"\n************************Main      Menu*************\n";
cout<<"\n==================================\n";

cout << "\n1. Insert an element\n2. Delete an element\n3. Display
the queue\n4. Exit\n";

        cout << "\nEnter your choice: ";

        cin >> choice;

        switch (choice) {

            case 1:

                insert();

                break;

            case 2:

                deleteItem();

                break;

            case 3:
```

```
                display();
                break;
            case 4:
                exit(0);
                break;
            default:
                cout << "\nEnter a valid choice\n";
                getch(); // Wait for a key press to continue
        }
    }
}

void insert() {
    int item;
    clrscr();
    cout << "\nEnter the element: ";
    cin >> item;
    if (rear == maxsize - 1) {
        clrscr();
        cout << "\nOVERFLOW\n";
        getch();
        return;
    }
```

```
    if (front == -1 && rear == -1) {
        front = 0;
        rear = 0;
    } else {
        rear = rear + 1;
    }
    queue[rear] = item;
    cout << "\nValue inserted\n";
    getch();
}

void deleteItem() {
    int item;
    clrscr();
    if (front == -1 || front > rear) {
        cout << "\nUNDERFLOW\n";
        getch();
        return;
    } else {
        item = queue[front];
        if (front == rear) {
            front = -1;
            rear = -1;
```

```
        } else {
            front = front + 1;
        }
        cout << "\nValue deleted\n";
        getch();
    }
}

void display() {
    int i;
    clrscr();
    if (rear == -1) {
        cout << "\nEmpty queue\n";
        getch();
    } else {
        cout << "\nPrinting values...\n";
        for (i = front; i <= rear; i++) {
            cout << queue[i] << endl;
        }
        getch();
    }
}
```

Output:

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

==========

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?1

Enter the element

123

Value inserted

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

==========

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ? 1

Enter the element

90

Value inserted

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

==========

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?2

value deleted

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

==========

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?3

printing values .....

90

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

==========

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

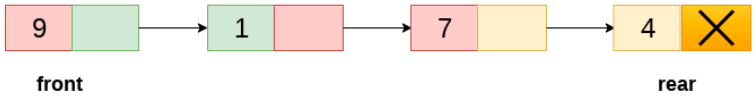
Enter your choice ?4

2. Linked Representation of Queue:

In the linked queue, two memory pointers are maintained, specifically the front pointer and the rear pointer. The front pointer stores the address of the initial element in the queue, while the rear pointer z` stores the address of the last element in the queue.

Insertions and deletions are executed at the rear and front ends, respectively. When both the front and rear pointers are NULL, it signifies that the queue is empty.

The linked representation of the queue is illustrated in the figure below:



Linked Queue

The storage requirement of linked representation of a queue with n elements is o(n) while the time requirement for operations is o(1).

Operations carried out on a queue The primary operations that can be executed on a queue are outlined as follows:

I. Enqueue: Enqueueing is employed to add an element to the rear end of the queue. It returns void.

II. Dequeue: Dequeuing involves removing an element from the front-end of the queue. It also returns the element removed from the front-end as an integer value.

III. Peek: This third operation returns the element pointed to by the front pointer in the queue but doesn't remove it.

IV. Queue Overflow (isFull): This operation indicates an overflow condition when the queue is entirely full.

V. Queue Underflow (isEmpty): This operation indicates an underflow condition when the queue is empty, meaning no elements are present in the queue.

4 D- queue

A deque is short for "Double Ended Queue." It's a type of data structure where you can add or remove things from both ends. You can think of it as a fancier version of a regular queue.

But here's the twist: In a deque, you don't always follow the "First In, First Out" (FIFO) rule like a typical queue.

Here's how we show a deque: [picture of a deque].

In simpler terms, a deque is a special way of organizing and managing data where you can add or remove items from both the front and the back, and it doesn't always follow the usual order you might expect

Representation of deque

4.1 Types of deque

There are two types of deque -

- Input restricted queue
- Output restricted queue

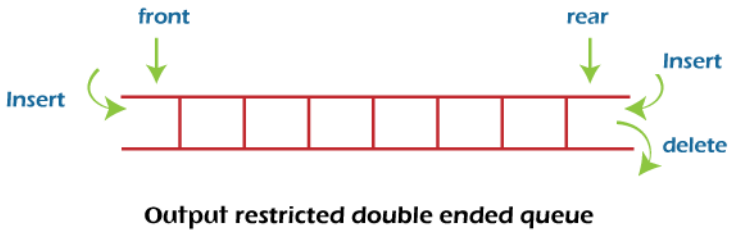
Input restricted Queue

In input restricted queue, insertion operation can be performed at only one end, while deletion can be performed from both ends.

input restricted double ended queue

Output restricted Queue

In output restricted queue, deletion operation can be performed at only one end, while insertion can be performed from both ends.



Output restricted double ended queue

5. **Priority Queue:**

A priority queue is a data structure that organizes elements, each having an associated priority. Elements with higher priorities are dequeued (removed) before elements with lower priorities. It's like a line of people where the person with the highest priority gets served first.

Example: Imagine you have a to-do list, and each task has a priority level (e.g., high, medium, low). A priority queue can help you manage your tasks, ensuring you tackle high-priority tasks before lower-priority ones.

Properties of Priority Queue:

1. **Priority Order:** Elements are dequeued in order of their priority, with higher-priority elements coming first.
2. **Fast Insertion:** Inserting elements into a priority queue is usually efficient.
3. **Fast Retrieval:** Getting the highest-priority element is quick.
4. **No Particular Order:** Elements within the same priority level may not have a specific order.

Types of Priority Queue:

1. **Min Priority Queue:** In this type, elements with the lowest priority (e.g., smallest value) have the highest priority.
2. **Max Priority Queue:** Here, elements with the highest priority (e.g., largest value) are dequeued first.

Example: If you have a min priority queue, a task with a priority of 1 will be dequeued before a task with a priority of 2. In a max priority queue, it's the opposite - a task with a priority of 5 will be dequeued before a task with a priority of 2.

In conclusion, a priority queue is a useful data structure for managing tasks or elements based on their priorities. It ensures that the highest-priority tasks are processed first, making it a valuable tool for various applications, such as job scheduling, network routing, or data compression