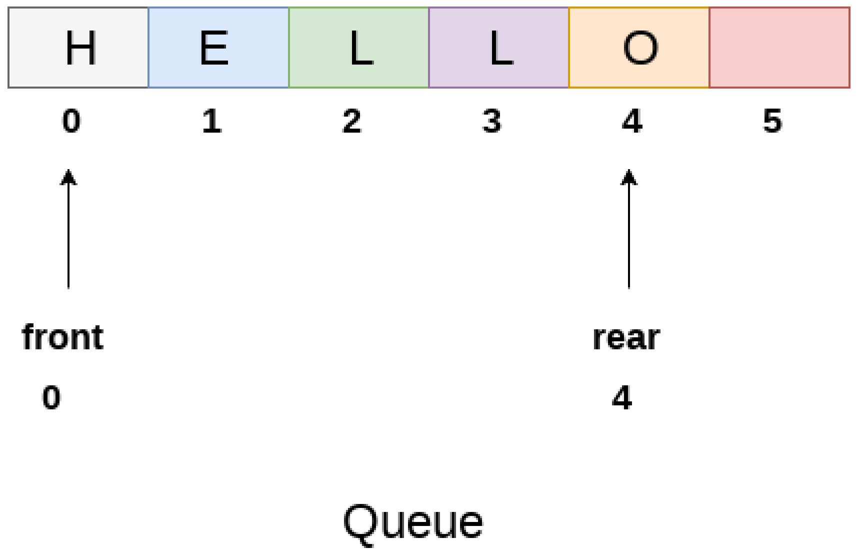
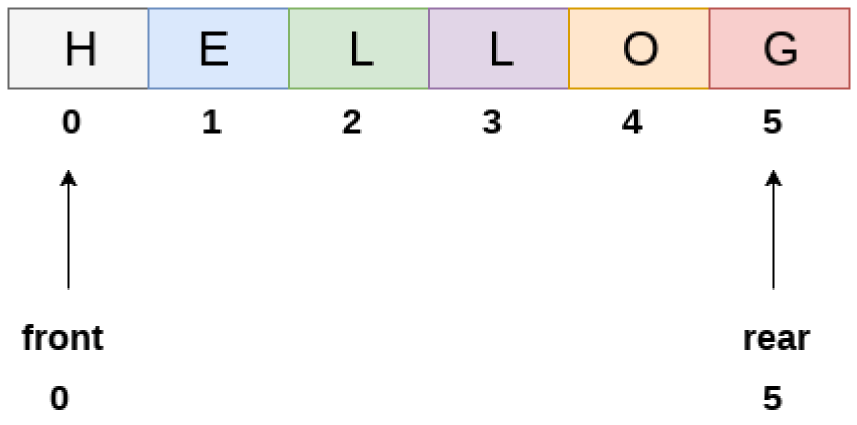
Array representation of Queue

We can easily represent queue by using linear arrays. There are two variables i.e. front and rear, that are implemented in the case of every queue. Front and rear variables point to the position from where insertions and deletions are performed in a queue. Initially, the value of front and queue is -1 which represents an empty queue. Array representation of a queue containing 5 elements along with the respective values of front and rear, is shown in the following figure.

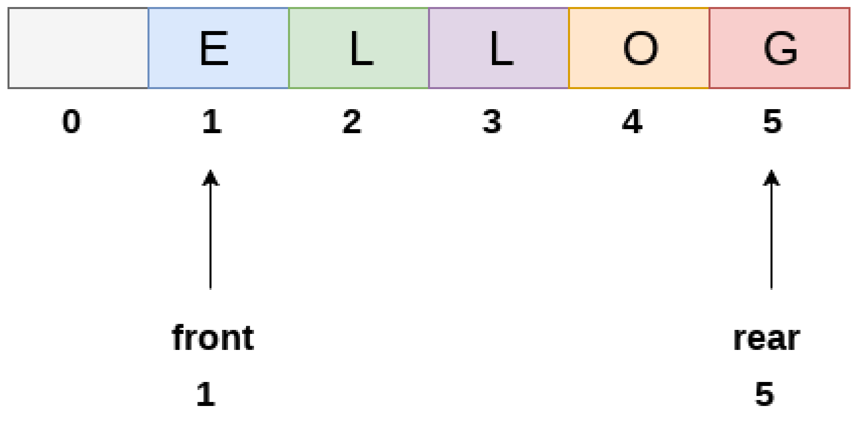


The above figure shows the queue of characters forming the English word **"HELLO"**. Since, No deletion is performed in the queue till now, therefore the value of front remains -1 . However, the value of rear increases by one every time an insertion is performed in the queue. After inserting an element 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

```
#include <bits/stdc++.h>
using namespace std;

struct Queue {
    int front, rear, capacity;
    int* queue;
    Queue(int c)
    {
        front = rear = 0;
        capacity = c;
        queue = new int;
    }

    ~Queue() { delete[] queue; }

    // function to insert an element
    // at the rear of the queue
    void queueEnqueue(int data)
```

```
{
    // check queue is full or not
    if (capacity == rear) {
        printf("\nQueue is full\n");
        return;
    }

    // insert element at the rear
    else {
        queue[rear] = data;
        rear++;
    }
    return;
}

// function to delete an element
// from the front of the queue
void queueDequeue()
{
    // if queue is empty
    if (front == rear) {
        printf("\nQueue is  empty\n");
        return;
    }

    // shift all the elements from index 2 till rear
    // to the left by one
    else {
        for (int i = 0; i < rear - 1; i++) {
            queue[i] = queue[i + 1];
        }

        // decrement rear
        rear--;
    }
    return;
}

// print queue elements
void queueDisplay()
{
    int i;
    if (front == rear) {
        printf("\nQueue is Empty\n");
        return;
    }

    // traverse front to rear and print elements
    for (i = front; i < rear; i++) {
        printf(" %d <-- ", queue[i]);
    }
    return;
}

// print front of queue
void queueFront()
{
    if (front == rear) {
        printf("\nQueue is Empty\n");
        return;
    }
    printf("\nFront Element is: %d", queue[front]);
```

```
        return;
    }
};

// Driver code
int main(void)
{
    // Create a queue of capacity 4
    Queue q(4);

    // print Queue elements
    q.queueDisplay();

    // inserting elements in the queue
    q.queueEnqueue(20);
    q.queueEnqueue(30);
    q.queueEnqueue(40);
    q.queueEnqueue(50);

    // print Queue elements
    q.queueDisplay();

    // insert element in the queue
    q.queueEnqueue(60);

    // print Queue elements
    q.queueDisplay();

    q.queueDequeue();
    q.queueDequeue();

    printf("\n\nafter two node deletion\n\n");

    // print Queue elements
    q.queueDisplay();

    // print front of the queue
    q.queueFront();

    return 0;
}
```

Output

```
Queue is Empty

 20 <--  30 <--  40 <--  50 <--

Queue is full

 20 <--  30 <--  40 <--  50 <--


after two node deletion



 40 <--  50 <--

Front Element is: 40
```

Time Complexity: O(1) for Enqueue (element insertion in the queue) as we simply increment pointer and put value in array and O(N) for Dequeue (element removing from the queue) as we use for loop to implement that.
Auxiliary Space: O(N), as here we are using an N size array for implementing Queue

Applications of a Queue:

The queue is used when things don’ t have to be processed immediately, but have to be processed in First In First Out order like Breadth First Search. This property of Queue makes it also useful in the following kind of scenarios.

1. When a resource is shared among multiple consumers. Examples include CPU scheduling, Disk Scheduling.
2. When data is transferred asynchronously (data not necessarily received at the same rate as sent) between two processes. Examples include IO Buffers, pipes, file IO, etc.
3. Linear Queue: A linear queue is a type of queue where data elements are added to the end of the queue and removed from the front of the queue. Linear queues are used in applications where data elements need to be processed in the order in which they are received. Examples include printer queues and message queues.
4. Circular Queue: A circular queue is similar to a linear queue, but the end of the queue is connected to the front of the queue. This allows for efficient use of space in memory and can improve performance. Circular queues are used in applications where the data elements need to be processed in a circular fashion. Examples include CPU scheduling and memory management.
5. Priority Queue: A priority queue is a type of queue where each element is assigned a priority level. Elements with higher priority levels are processed before elements with lower priority levels. Priority queues are used in applications where certain tasks or data elements need to be processed with higher priority. Examples include operating system task scheduling and network packet scheduling.
6. Double-ended Queue: A double-ended queue, also known as a deque, is a type of queue where elements can be added or removed from either end of the queue. This allows for more flexibility in data processing and can be used in applications where elements need to be processed in multiple directions. Examples include job scheduling and searching algorithms.
7. Concurrent Queue: A concurrent queue is a type of queue that is designed to handle multiple threads accessing the queue simultaneously. Concurrent queues are used in multi-threaded applications where data needs to be shared between threads in a thread-safe manner. Examples include database transactions and web server requests.