CS-2001
Data Structures
Fall 2022
Queue ADT – Linked List Implementation

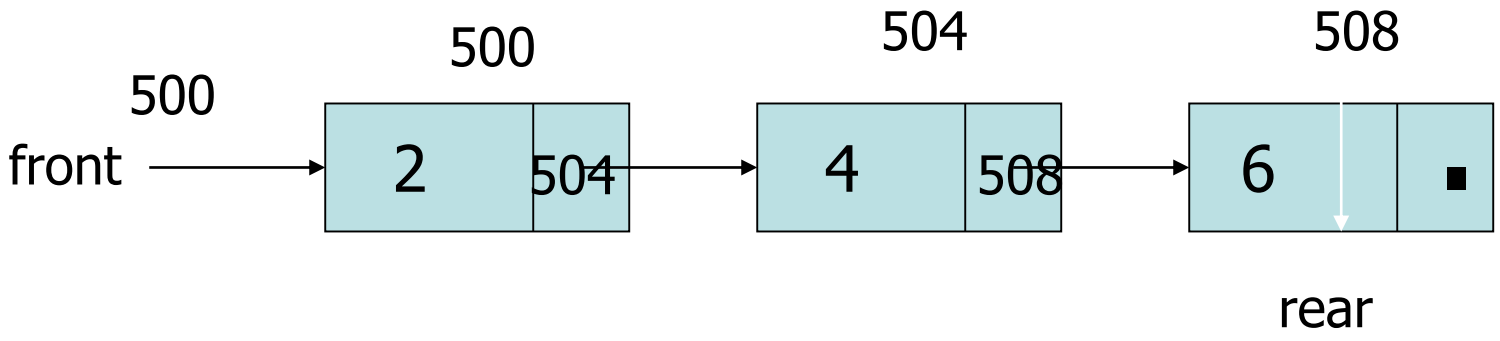
Mr. Muhammad Yousaf
National University of Computer and Emerging Sciences,
Faisalabad, Pakistan.

Pointer-based Implementation

Pointer-Based Implementation (Singly Link List)

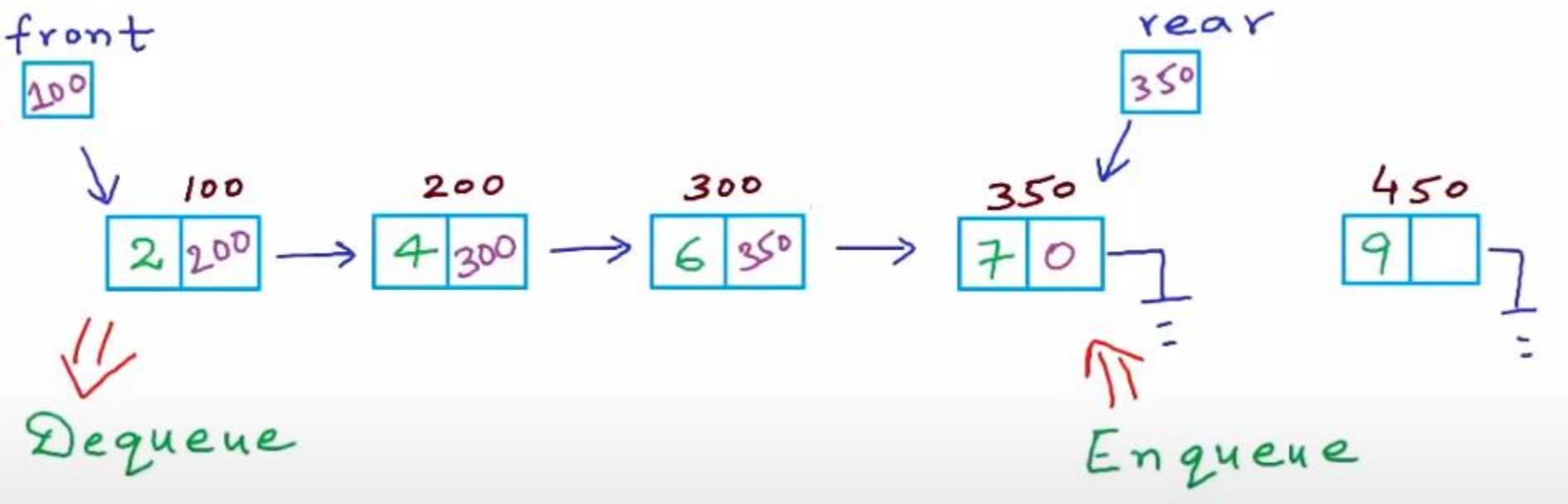
- Queue Class maintains two pointers
  - `front`: A pointer to the first element of the queue
  - `rear`: A pointer to the last element of the queue

  - Insertion: Cost of insertion can be O(n)
  - Deletion: Cost of deletion can be O(1)



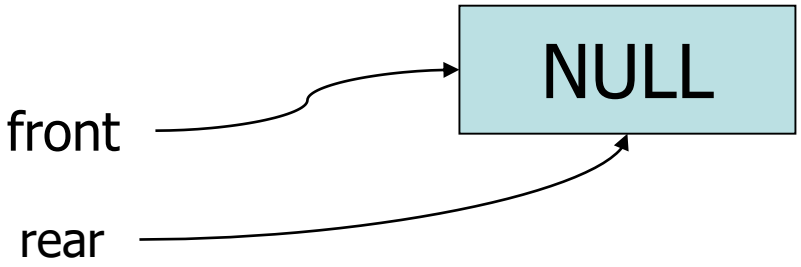
Pointer-Based Implementation of Queues

- Queue Class maintains two pointers
  - `front`: A pointer to the first element of the queue
  - `rear`: A pointer to the last element of the queue

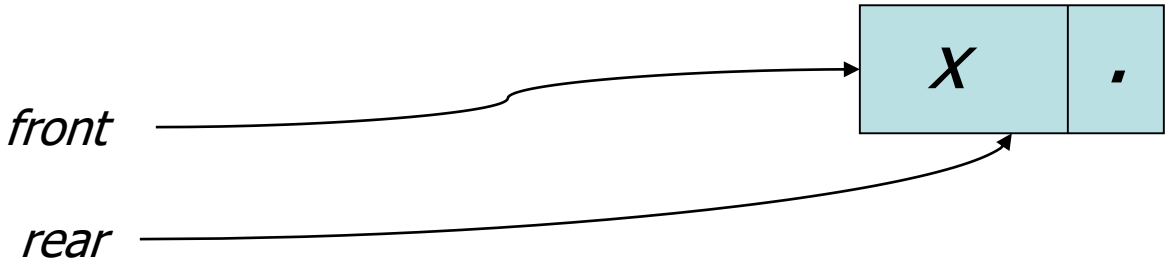


Queue Operations (1)

- MAKENULL(Q)

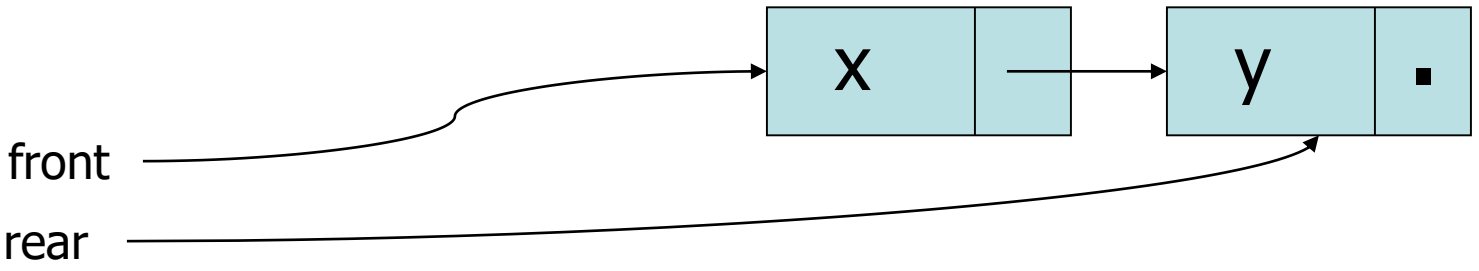


- ENQUEUE (x, Q)

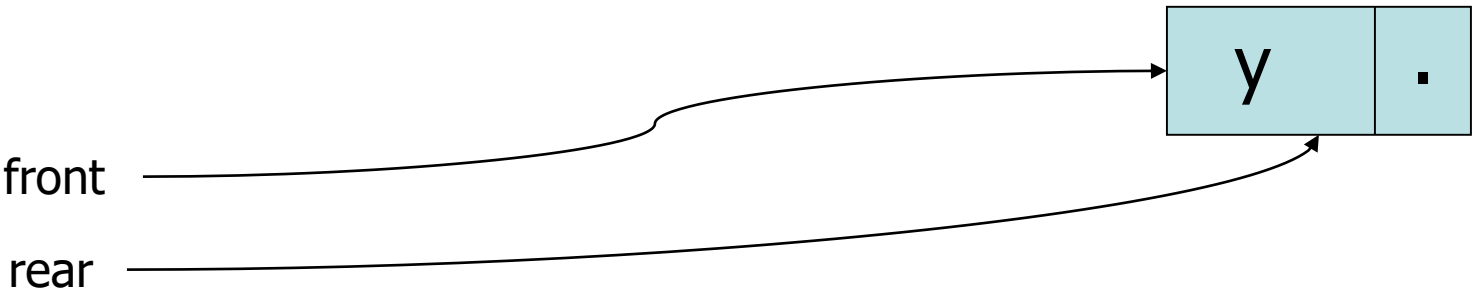


Queue Operations

- ENQUEUE(y, Q)



- DEQUEUE (Q)



Pointer Implementation – Code (1)

```
class Queue
{
   private:
      struct node  // Structure to defined linked list node
      {
         int value;
         node *next;
      };
      node *front; // pointer to the first node
      node *rear;  // pointer to the last node
      int numItems;     // Number of nodes in the linked list
   public:
      Queue();
      ~Queue();
      bool isEmpty();
      void enqueue(int);
      bool dequeue(int &);
      void makeNull();
};
```

Pointer Implementation – Code (2)

- Constructor

```
Queue::Queue()
{
   front = NULL;
   rear = NULL;
   numItems = 0;
}
```

- `isEmpty()` returns true if the queue is full and false otherwise

```
bool Queue::isEmpty()
{
   if (numItems == 0)
      return true;
   else
      return false;
}
```

Pointer Implementation – Code (3)

- Function `enqueue` inserts the value in `num` at the end of the Queue

```
void Queue::enqueue(int num)
{
    node *temp;
    temp = new node;
    temp->value = num;
    temp->next = NULL;
    if (isEmpty()) {
        front = temp;
        rear = temp;
    }
    else {
        rear->next = temp;
        rear = temp;
    }
    numItems++;
}
```

Pointer Implementation – Code (4)

- Function `dequeue` removes and returns the value at the front of the Queue

```
bool Queue::dequeue(int& num)
{
    node *temp;
    if (isEmpty())
    {
        cout << "The queue is empty.\n";
        return false;
    }

    num = front->value;
    temp = front->next;
    delete front;
    front = temp;
    numItems--;

    return true;
}
```

Pointer Implementation – Code (5)

- Destructor

```
Queue::~Queue()
{
   makeNull();
}
```

- `makeNull()` resets `front` & `rear` pointers to `NULL` and sets `numItems` to 0

```
void Queue::makeNull()
{
   int x;
   while(!isEmpty()){
      dequeue(x);
   }
}
```

Using Queues

```
int main()
{
   Queue iQueue;

   cout << "Enqueuing 5 items...\n";
   // Enqueue 5 items
   for (int x = 0; x < 5; x++)
      iQueue.enqueue(x);

   // Deqeue and retrieve all items in the queue
   cout << "The values in the queue were:\n";
   while (!iQueue.isEmpty())
   {
      int value;
      iQueue.dequeue(value);
      cout << value << endl;
   }
   return 0;
}
```

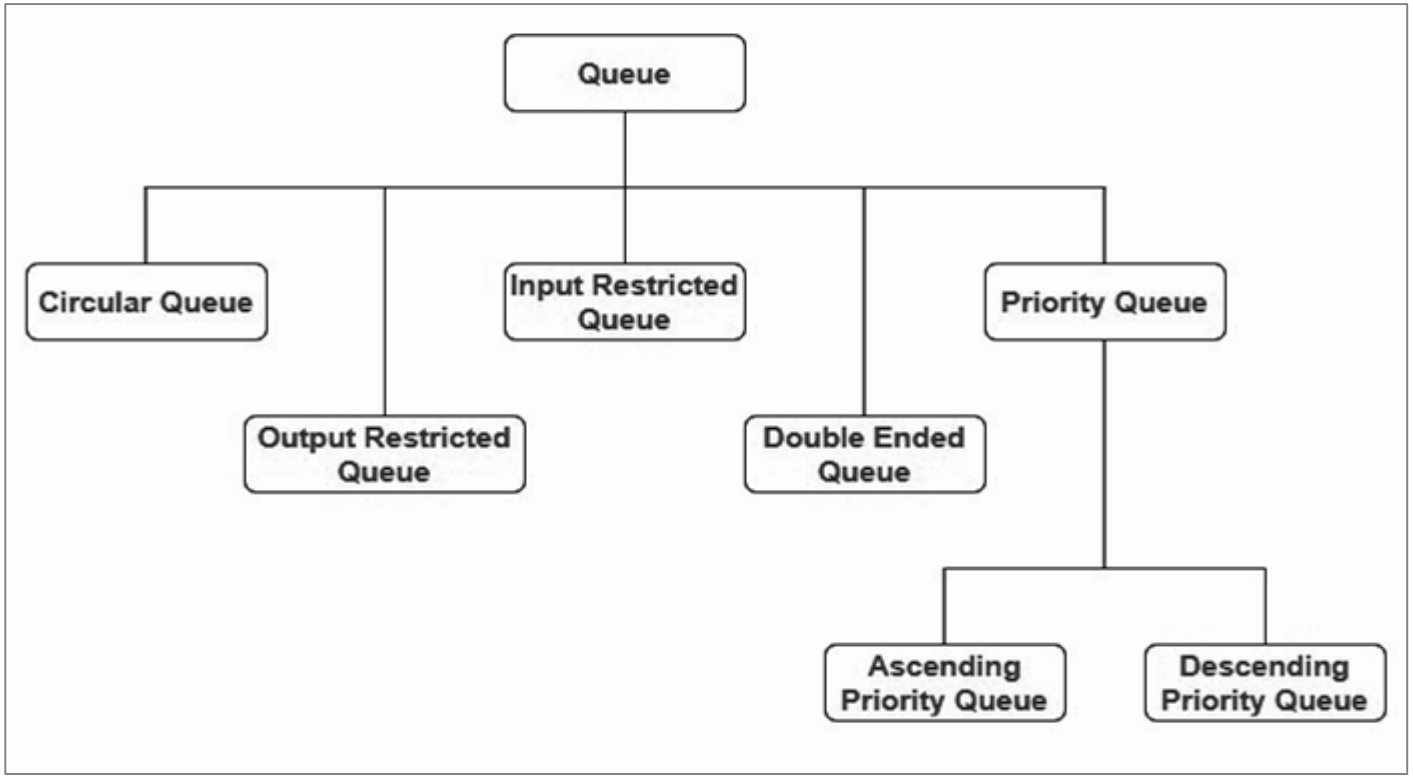
Output:

```
Enqueuing 5 items...
The values in the queue were:
0
1
2
3
4
```

Variations of Queues

Types of Queues

Following types of queues can be implemented;



Circular Queue

- **Circular Queue** is a linear data structure in which the operations are performed based on FIFO principle and the last position is connected back to the first position to make a circle. It is also called 'Ring Buffer'

- **Memory Management:** The unused memory locations in the case of ordinary queues can be utilized in circular queues.

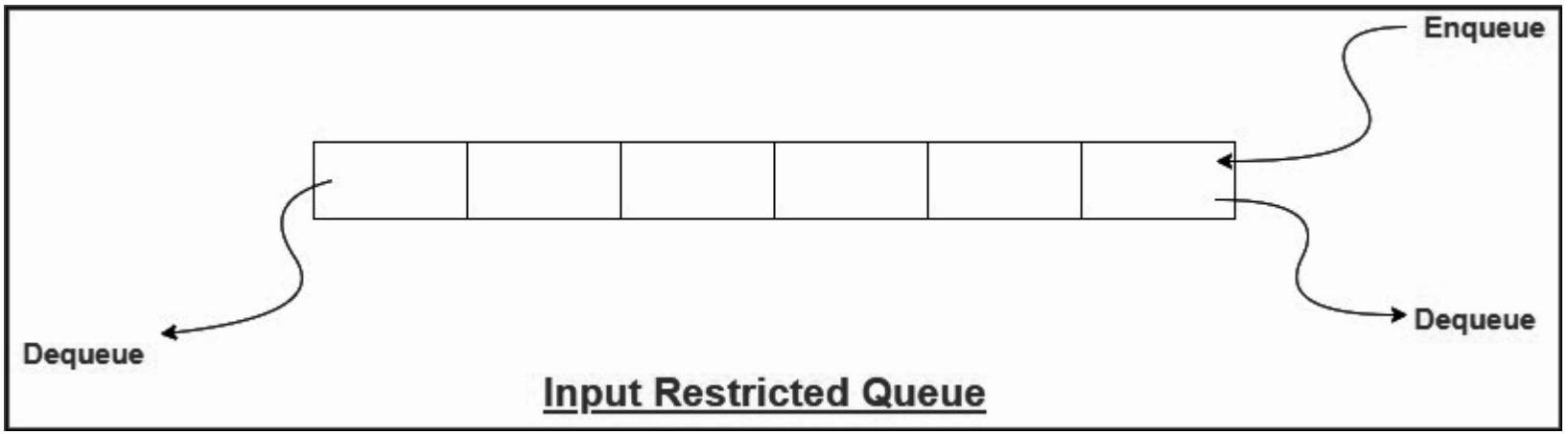
- **Traffic system:** In a computer-controlled traffic system, circular queues are used to switch on the traffic lights one by one repeatedly as per the time set.

- **CPU Scheduling:** Operating systems often maintain a queue of processes that are ready to execute or that are waiting for a particular event to occur.

Input Restricted Queue

- In this type of Queue, the Insertion can be taken from one side only(rear) and deletion of elements can be done from both sides(front and rear). This kind of Queue does not follow FIFO(first in first out).

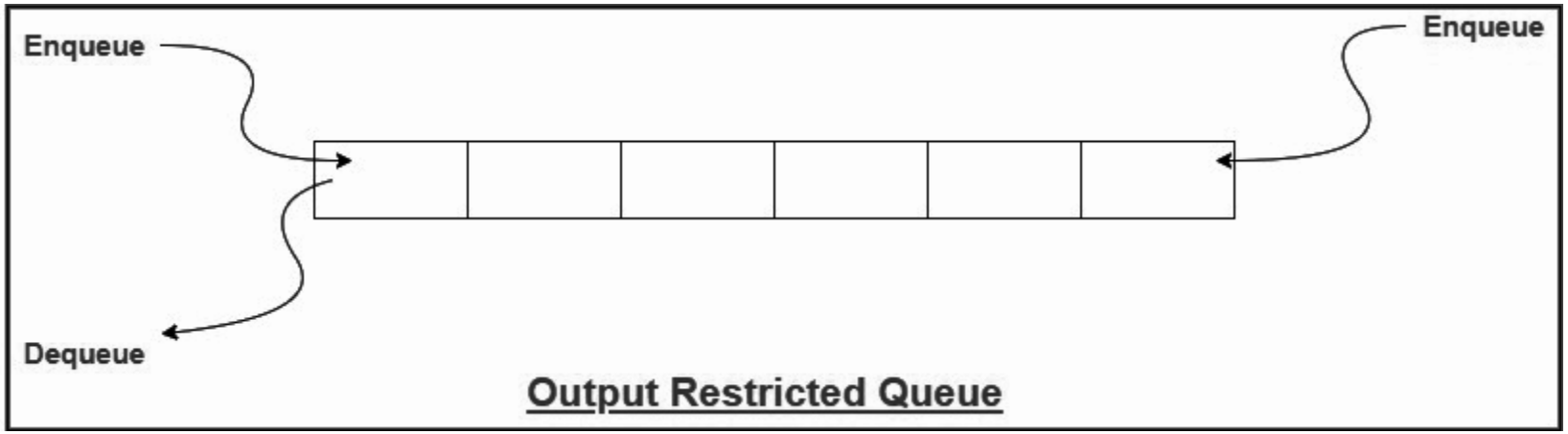
- This queue is used in cases where, if there is a need to remove the recently inserted data for some reason and one such case can be irrelevant data, performance issue, etc.



Output Restricted Queue

- In this type of Queue, the Insertion can be done from both sides(rear and front) and the deletion of the element can be done from only one side(front).

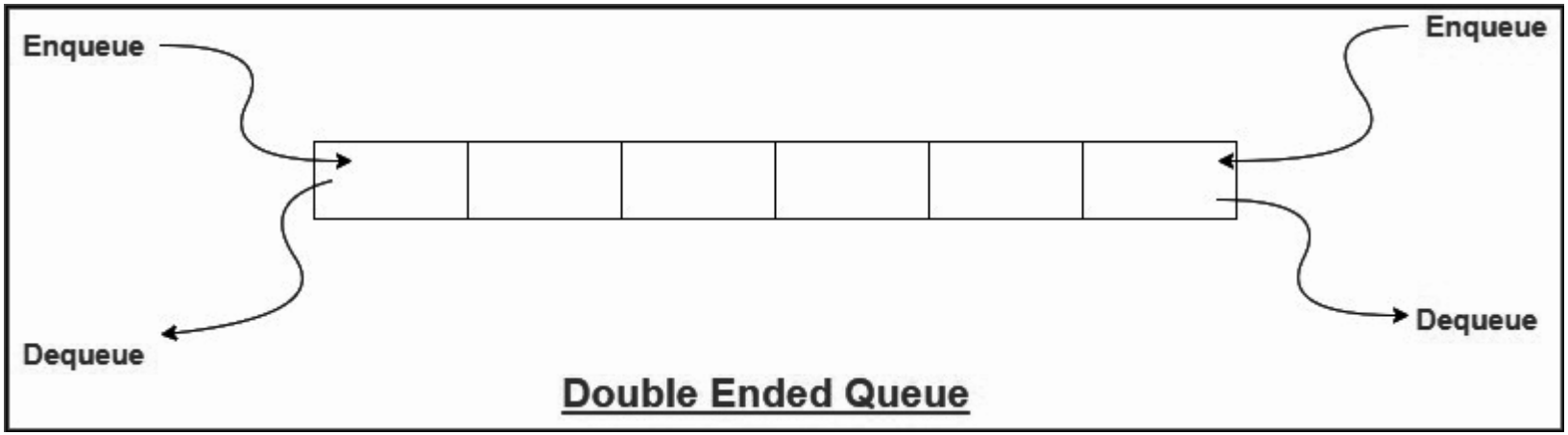
- This queue is used in the case where the inputs have some priority order to be executed and the input can be placed even in the first place so that it is executed first.



Double Ended Queue (Deque)

- Double Ended Queue is also a Queue data structure in which the insertion and deletion operations are performed at both the ends (front and rear). That means, we can insert at both front and rear positions and can delete from both front and rear positions.

- **Deque** supports both stack and queue operations.
- It can be useful in certain applications, where elements need to be removed and or added both ends can be efficiently solved using **Deque**.



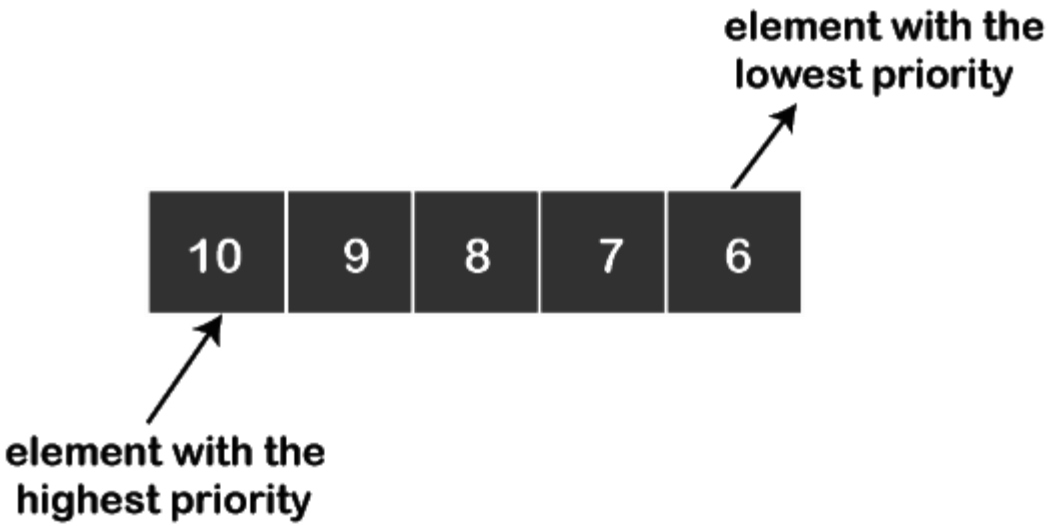
Double Ended Queue (Deque)

```
//Deque ADT
class dequeue {
   node*front, *read;
   public:
      dequeue();
      void insert_at_beg(int);
      void insert_at_end(int);
      void delete_fr_front();
      void delete_fr_rear();
      int getFront();
      int getLast();
      bool isEmpty();
};
```

Priority Queues

- A priority queue is a special type of queue in which each element is associated with a priority and is served according to its priority.

- There are two types of Priority Queues.
  - Ascending priority
  - Descending priority



Ascending Priority Queue (Max priority)

- Element can be inserted arbitrarily/ but only smallest element can be removed.

- For example, suppose there is an array having elements 4,2,6,1,8 in the same order. So, while insertion, the elements will be in added in the same sequence but while deleting, the order will be 8,6,4,2,1.

Descending Priority Queue (Min priority)

- Element can be inserted arbitrarily but only the largest element can be removed first from the given Queue.

- For example, suppose there is an array having elements 4, 2, 6,1,8 in the same order. So, while inserting the elements, the insertion will be in the same sequence but while deleting, the order will be 1,2,4,6,8.

Any Question So Far?