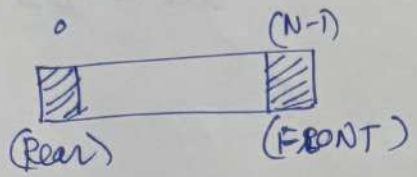
Queue:
(FIFO principle)

a linear datastructure that follows FIFO (first in first out) principle, where, insertion happens at REAR and deletion happens from the FRONT;



→ Queue is used when "order of process matters".

Operations:

1. enqueue → insert
2. dequeue → deletes
3. front → deleting variable
4. rear → insertion variable
5. isempty → check if empty
6. isfull → checks if full;

Array implementation:

```
class queue {
  public:
    int *arr;
    int rear;
    int front;
    int size;
};

queue(int n)
{
  arr = new int[size];
  size = n;
  rear = front = -1;
}
```

① Enqueue (int n)

```
{
  if(rear == n-1)
    return false;
  else
    arr[rear++] = n;
```

③ dequeue()

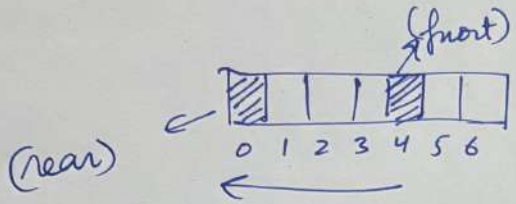
```
if (front == -1) || (front > rear)
    cout << "empty";
else
    front++;
```

② front()

```
return arr[front];
```

④ Display()

```
for(int i = front; i < rear; i++)
    cout << queue[i];
```



Problem with simple queue:

① space waste after dequeue:, after deletion, space remains unused in the queue;

Solution → circular queue

features:

① FIFO order preserved
② order preserved
③ fair scheduling
④ efficient for sequential processing
⑤ support BFS traversal;

Advantages

① fairness (No starvation)
② simple logic
③ best for scheduling / fairness
④ useful in Real time system

Disadvantages

① No random access
② fixed size array
③ search is slow (O(n))

Types of Queeue:

① Simple ② Circular ③ Deque ④ Priority

used in: ① print queue
② Ready queue
③ Customer queue
④ Node processing
⑤ Dequeue. (SW)

Keywords ① "First come First serve" ③ "level by level"
② "Process in order" ④ "Nearest / shortest"
⑤ "Sliding window" ⑥ "Multiple sources" ⑦ "Minimum steps/distance"

Stack	queue
① LIFO	FIFO
② DFS	BFS
③ Backtracking	Scheduling
④ Undo/ Redo	Task queue

Queue in BFS:
→ " to explore nodes level by level"

Enqueue → O(1)
dequeue → O(1)

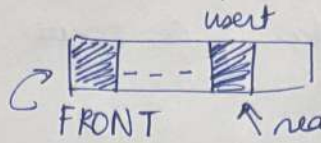
☆ Queue is preferred when order of processing is important; } IMP
☆ BFS uses queue for level wise traversal
☆ Circular queue solves space wastage problem

Circular queue: Dequeue K bad ki baki space to use krta hai;

→ Circular queue where the last position is connected to the first position to efficiently utilize memory

Circular queue:

```
class queue {
    int *arr;
    int front;
    int rear;
    int size;
};
```



* (New element always in rear)
* (FRONT s element delete hoga)

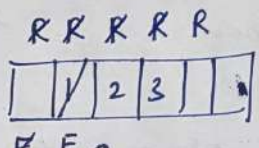
```
queue(int n)
{
    size = n;
    arr = int *arr[size];
    front = rear = -1;
}
```

```
bool enqueue(int n)
{
    if (isfull()) return false;
    else {
        rear = (
    if (front == -1) front = 0;
    else rear = (rear+1) % front.size;
      arr[rear] = value;
      return true;
}
```

```
bool dequeue()
{
    if (isempty()) return false;
    if (front == rear)
        return false;
        front = rear = -1;
    }
    else {
        front = (front+1) % size;
    }
    return false;
```

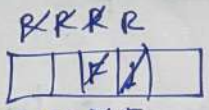
$$\begin{bmatrix} F = (1+1) \% 6 \\ F = 2 \% 6 \\ F = 2 \end{bmatrix}$$

(I)



(normal deletion)

(II)



front == rear
front = rear = -1;

* (last element deletion)

(I)

Queue template:

```
queue< pair <int, int>> q;
q.push (x, y);
while (!q.empty())
{
    auto curr = q.front();
    q.pop()
    for(auto dir : directions )   } // push neighbours.
    {
        //q.push (dir);
    }
}
```

Dequeue: linear DS, where insertion / deletion are allowed from both the ends.

(front & rear).

delete ↓ ↓ insert (back);

① FRONT SE DAAL SAKTE HAI!!
② REAR SE NIKAL SAKTE HAI!!

Queue	Dequeue
insert Rear	both
delete Front	Both
FIFO	flexible

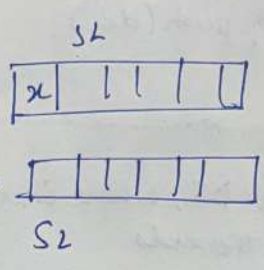
Uses:
① Sliding window
② Max/Min Nikalna
③ Monotonic maintain krna sequence

FIFO banane ke liye 2 stacks use hote hai!

→ Stack (in) 1st
Stack (out) 2nd.

enqueue → push in Stack(1)
dequeue → Agar stack 2 empty hai → stack 1 ke sare elements st 2 m daal
→ stack 2 ka pop();

```
void push(int x)
{
    s1.push(x)
}
```



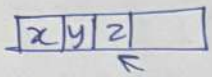

```
int pop()
{
    if(s2.empty())
    {
        while(!s1.empty())
        {
            s2.push(s1.top());
            s1.pop();
        }
    }
    int ans1 = s2.top();
}
```

// basically reversed order maintain kiye ke liye.

```
int peek()
{
    if(s2.empty())
    {
        while(!s2.empty())
```

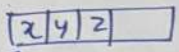
Ab queue kese pata / banega using stack;

Stack → LIFO



s.top() → (2).

queue → FIFO

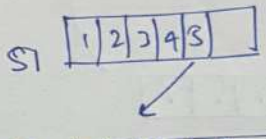


q.top() ⇒ (x)

So, 2 maintai proper insert/delete from both end we use 2 stack.

```
Stack <int> S1 ;
Stack <int> S2;

int push(int x)
{
   S1.push(x);
}
int pop()
{
   if(S2.empty())
   {
      while(!S1.empty())
      {
         // reverse the order
         S2.push(S1.top())
         S1.pop();
      }
   return S2.top();
}
```



queue.pop/ dequeue dega (1)

but stack k dega (3)

→ to 2 stack use karke change the order

pop → top ⇒ (1)