13/ April/22

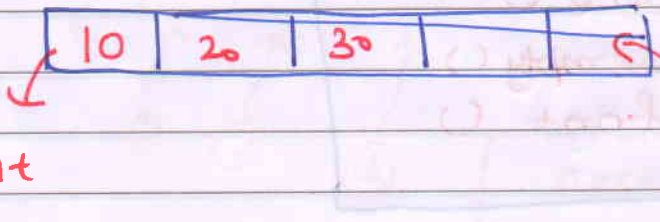
DATE

Queue → Abstract datatype

↓

First in first out

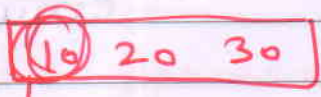
"FIFO"



"Jis order Mei element aaye usi order Mei bahar aa jaye."

Example → You go to buy Movie ticket if you are the first one to purchase ticket you will see the Movie then second person in the zone will get chance and in the last, the last person will get the chance

1 Insert → enque (10)
enque (20)
enque (30)



2 Access the first Element → → 10 → front()

3 Delete → dequeue(); → 20

DATE

4 Size → size()

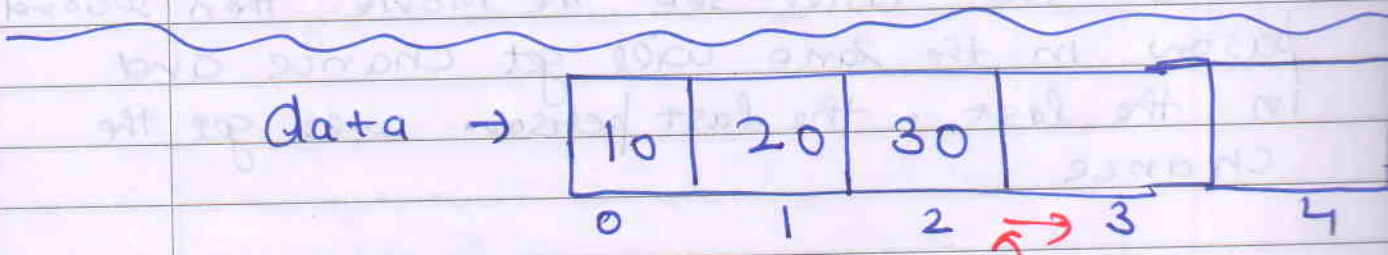
5 isempty() → T/F

enque()
deque()
size()
isempty()
front()

To Make Queue

↳ ① ARRAY

② Linked List



enque(10);
(20);
(30);

front(); → 10

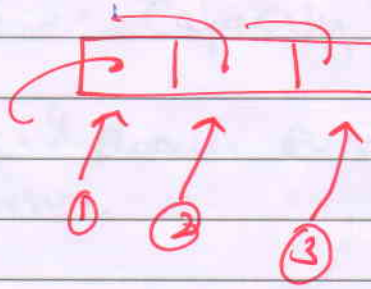
dequeue(); → 10
dequeue() → 20

PAGE

DATE

Insertion →

deletion →



next index

0
~~1~~
~~2~~
3

First index

~~-1~~
0

enque (10);
enque (20);
enque (30);

remains at

~~0~~
1

front () → 10

deque () → remove 10 and Make
First index = first index + 1

size () → 20 (next index - first index

is empty () → F

15/April

DATE

Queue Using ARRAYS (with template)

```
template < typename T >
Class Queue_Using array {

  T * data ;
int nextIndex ;
int firstIndex ;
int size ;
int capacity ;
Public :

Queue_using array (int S) {

  data = new T [S] ;

  nextIndex = 0 ; // initial element will be place
                  at 0 th position

  first element = -1 // once element will be place
                     in ARRAY we will increm
                     it to 0.

  Size = 0 ;
  Capacity = S ;
}

int getsize () {

  return size ;

}

bool isEmpty () {

    return size == 0 ;

}
```

classmate

PAGE

DATE

// insert element

```
Void enqueue (T element) {

~~data[~~

if ( size == capacity) {

cout << "Queue FULL" << endl;
return;

}

data [nextIndex ] = element ;
nextIndex = ( nextIndex + 1) % capacity ;
if (firstIndex == -1 ){
    firstIndex = 0 ;
}
    size ++ ;

}
    // Show value at 0th index.
T front () {

if ( firstIndex = -1 ) // or use isempty {

cout << "Queue is empty " ;
return 0 ;

}

Return data [ firstIndex ] ;

}
```

// Remove Element

```
T dequeue () {
if ( isempty () ) {
    cout << "queue is empty!" << endl;
    return 0;
}
    T ans = data [ first index ];
    first index = (first index + 1) % capacity;
    size --;
if ( size == 0 ) {
    first index = -1;
    next index = 0;
}
    return ans
}
```

```
# include < iostream >
using namespace std;
int Main () {
QueueusingArray <int> q (5);
q. Enqueue (10);
```

DATE

```
q.enqueue(20);
q.enqueue(30);
q.enqueue(40);
q.enqueue(50);
q.enqueue(60); → Queue full

cout << q.front() << endl; → 10
cout << q.dequeue() << endl; → 10
cout << q.dequeue() << endl; → 20
cout << q.dequeue() << endl; → 30

cout << q.getSize() << endl; → 2
cout << q.isEmpty() << endl; → 0
```

Output →

10	20	30	40	50

P.T.O

DATE

Dynamic Queues

```
Void enqueue ( T element ) {
if (size == capacity) {
T * newdata = new T [2 x capacity]
int j = 0
for ( int i = firstIndex ; i < capacity ; i++)
{ newdata [j] = data [i] ;
    j++ ;
}
for (int i = 0 ; i < first Index; i++) {
    newdata [j] = data [i] ;
    j++ ;
}
delete data [] ;
data = newdata ;
firstIndex = 0 ;
nextIndex = capacity ;
capacity * = 2 ;
}
data [nextIndex] = element
nextIndex = (nextIndex + 1) % capacity ;
```

```
if (first index == -1){
    first index = 6;
}
size ++;
}
```

Queue using linked list

Initially

head = NULL

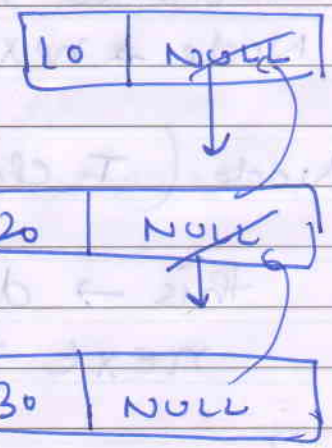
tail = NULL

size = 0

enqueue(10) →

enqueue (20)

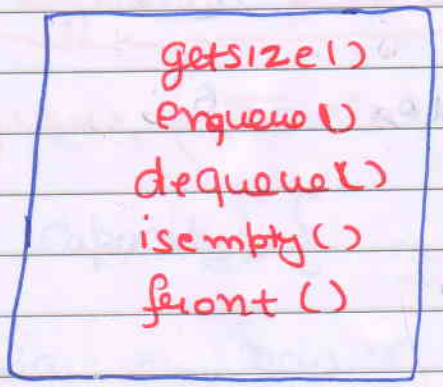
enqueue (30)



getSize() → 3

front → 10 (Head → next)

DATE



25/04/22

Queue

Code →

```
Template <typename T>
Class Node {
Public :
    T data ;                    // Node Class
    Node * next ;

    Node ( T element ) {
        this → data = data ;
        next = NULL ;
    }
} ;
```

DATE

```
Class Queue {

    Node<T> * head ;
    Node<T> * tail ;
     int size ;

Public :
                        // Constructor
    Queue ( ) {

        Head = NULL ; // Pushing from starting
        tail = NULL ; // poping from ending
         size = 0 ;
}

int getSize ( ) {

        return size ;
}

bool isempty ( ) {

        return size == 0 ;

}
```

DATE

```
// Inserting Element

Void enqueue ( T Element )
{
Node <T> * newnode = new Node <T> (element)

if ( head = = NULL)
{
    head = newnode ;
    Tail = newnode ;
}

 else {

     tail → next = newnode ;
      tail = newnode ; // tail = tail → next

}
  Size ++ ;

}

// Front element

if ( isempty () ) {

      return 0 ;
}
   return head → data ;
}
```

DATE

// delete Element

```
T dequeue () {
    if (isempty ()) {
        return 0;
    }
    T ans = head → data;        → saving data of head
    Node <T> * temp = head;
    head = head → next;
    delete temp;
    size --;
    return ans;
}
```

Since our Node is dynamic so we need to explicitly delete it so in order to delete first we stored its address into temp and then deleted temp;

#include<queue>

DATE

26/Apr/22

Inbuilt Queue

#include <queue>

enqueue(T element) → void push(T element)

T front() → T front()

T dequeue() → void pop();

int getsize() → int size()

bool isempty() → bool empty

"Made by us" "inbuilt"

Code →

```
#include<iostream>
#include <queue>
using namespace std;

int main() {

queue <int> q;

q.push(10);

q.push(30);
```

DATE

```
q.front()
    // Element at front.
Cout << q.front() << endl;
                // Removing top element.
q.pop()
                        // Size of Queue
Cout << q.size() << endl;

Cout << q.empty() << endl;

while( ! q.empty())
{
    Cout << q.front() << endl;
    q.pop();
}
```