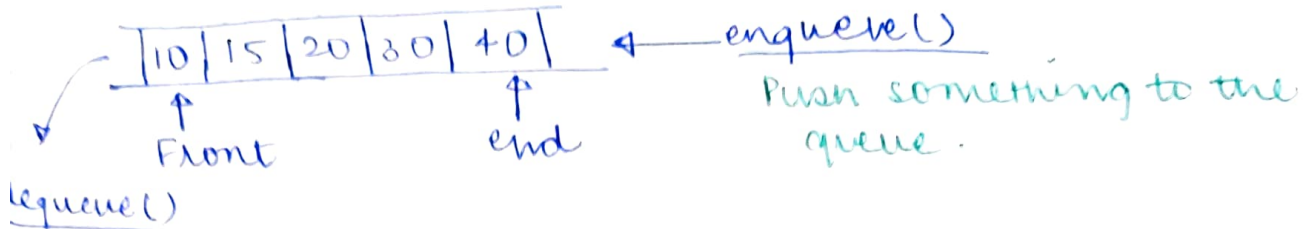


QUEUE

FIFO - first in first out
(Abstract DT)



operations

- enqueue(x) ← pushing x queue
- dequeue ← Removing element from back of queue
- getFront() ← get the element Front

get the element Rear ← getRear()

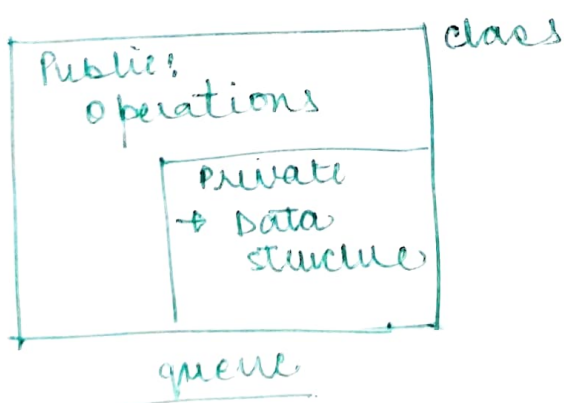
the item which is inserted last is called rear.

the item which is going to be removed next

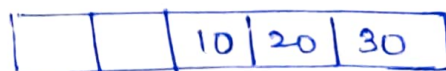
• size → returns the size of queue

• isEmpty() → returns true if queue is empty

Implementation



- We have to maintain two variables namely front and rear.
- Suppose after enqueue & dequeue operations, this is the status of queue



one way

shift all the elements by k spaces (k = number of free spaces at front).

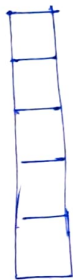
other way

We should try to implement queue in circular manner.

① In the 1st option - the complexity will be $O(N)$

② Implementing and option - updating indices in circular manner.

~~cap~~ cap = 5



enqueue (10)

enqueue (20)

dequeue ();

enqueue (40);

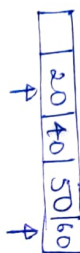
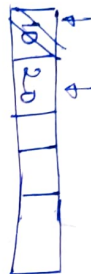
dequeue ();

enqueue (50)

enqueue (60)

enqueue (70)

A



(Error!) as the nextIndex has reached to cap-1

Ideally we should move nextIndex to 0 if (circularly)

Enqueue()

void enqueue (int el) {

if (size == cap)

cout << "queue is full";

return;

queue[nextIndex] = el;

nextIndex = (nextIndex + 1) % capacity;

if (firstIndex == -1)

firstIndex = 0;

size++;

}

enqueue()

int dequeue () {

if (isEmpty ()) {

cout << "stack is empty!"; << endl;

}

int res = queue[firstIndex];

firstIndex = (firstIndex + 1) % capacity;

size--;

if (size == 0) {

firstIndex = -1;

nextIndex = 0;

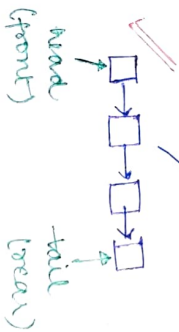
}

return res;

}

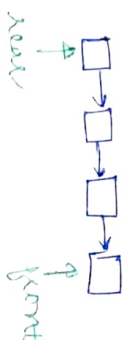
Using linked list

We have two choices



- Insertion at rear $O(1)$
- deletion at front $O(1)$

(better choice)



- Insertion of rear $O(1)$
- + deletion at front $O(N)$

Queue in STL

* queue <datatype> queueName;

int, char, float...

Operations

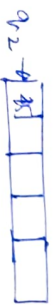
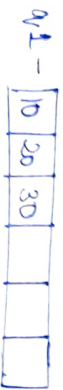
- ① q.front() → Returns the element which is going to be popped next
- ② q.back() → Returns the element which is inserted at last
- ③ q.pop() → front will be deleted (dequeue)
- ④ q.push() → adds an item at end (enqueue)
- ⑤ q.size() → Returns size of queue
- ⑥ empty() → Returns if

Internally queue is a container adaptor which uses dequeue. stack also uses dequeue if not specified

Implement stack using queue

We will use an auxiliary queue for the purpose

We want to maintain life patterns so any element must be inserted in the front of queue.



enqueue (45)



More variations

- ① Implement stack using ~~array~~ queue by making pop operation costly.
- ② Implement stack using 1 queue (using recursion call stack).
- ③ Implement queue using stack.

Reversing a Queue

I/P → q = {10, 5, 15, 20} O/P → {20, 15, 5, 10}

↑
↑
↑
↑

front
rear
front
rear

- ① put items of queue in the stack and then pop (it will reverse the order)

~~void reverse (queue <int> q) {~~

stack <int> s;

while (q.empty() == false) {

 s.push (q.front());

 q.pop();

}

while (s.empty() == false) {

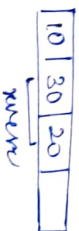
 q.push (s.top());

 s.pop();

}

③ Recursive

suppose we have reversed n-1th queue



now dequeue 10 & enqueue it

20 20 10 → Reverse

void reverse (queue <int> q) {

if (q.empty() == true)

return;

int x = q.top();

q.pop();

reverse(q);

q.push(x);

Recursively reverses
a queue;

}

Generate Numbers with given Digit

digits → {5, 6}

numbers → 5, 6, 55, 56, 65, 66, 555, 556, ...

we can use recursive method + queue,

void printFuntN (int n) {

queue <int> q;

q.push('5');

q.push('6');

for (int i = 0; i < count; i++) {

string cur = q.top();

cout << cur;

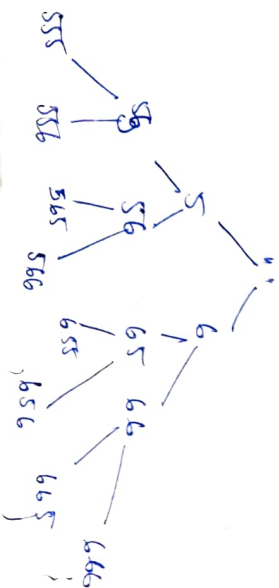
q.pop();

q.push (cur + '5');

q.push (cur + '6');

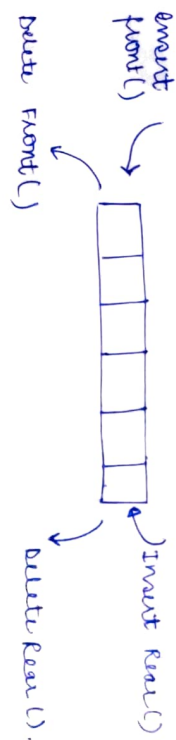
}

}



Deque

• Insertion and ~~delete~~ deletion at both ends.



Operations

① getFront() → Returns the front element (inserted first)

② getRear() → Returns the rear element (inserted last)

③ isFull() → True if queue is full.

④ isEmpty() → True if queue is empty;

⑤ size() → Returns size of queue.



Empty deque

① insertFront (10);

② insertFront (20);

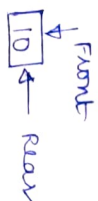
③ insertRear (30);

④ ————— (40);

⑤ insertFront (50);

⑥ deleteFront();

⑦ ————— Rear();



f → [20 | 10] ← Rear

f → [20 | 10 | 30] ← x

f → [20 | 10 | 30 | 40] ← x

f → [50 | 20 | 10 | 30 | 40] ← x

f → [20 | 10 | 30 | 40] ← x

f → [20 | 10 | 30] ← x

Implementations

→ linked list
(using doubly linked)
→ Array

$O(1)$
complexity

(circular array)

Applications

- ① Maintaining history of actions
- ② A task process scheduling algorithm
- ③ Implementing a priority queue with only two priorities

→ priority 1 items can be finished in only order but must be completed before priority 2.

- ④ Maximum/minimum of all subarrays of size 'k' in an array.

Deque in C++ STL

- allows random access

```
for (auto x : dq) {  
    cout << x;  
}
```

} prints queue

Operations

- (a) `dq.push_front(5);` → insertion at front
- (b) `dq.push_back(50);` → insertion at rear/end
- (c) `dq.front();`
- (d) `dq.back();`
- (e) `dq.begin();` → iterator pointing to first element
- (f) `dq.end();` → points to iterator beyond the last element

→ auto it = dq.begin();

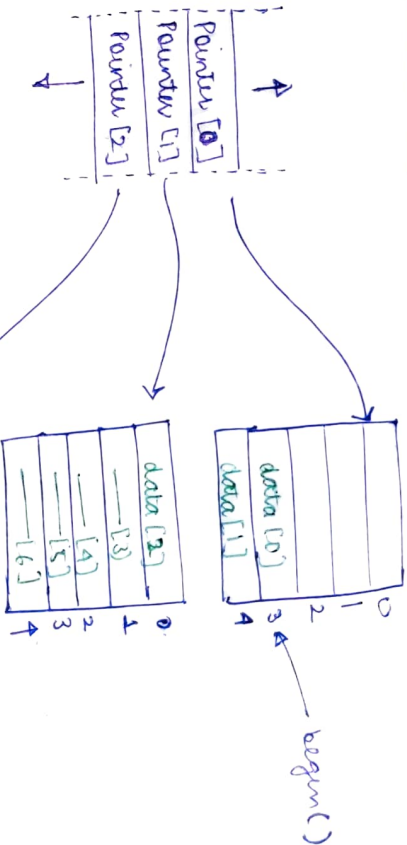
10 | 20 | 30
↑
it

→ `it++;`
→ `dq.insert(it, 25);` → inserts 25 before the element pointed by it

10 | 25 | 20 | 30

→ `dq.pop_front();` → removes element from front
→ `dq.pop_back();` → removes element from rear

How does deque work



• we start from the middle of pointers array

• we start from the middle of the data chunk and on next first we insert data from the middle in the upward direction

For instance

`data[4]` is inserted first
`data[3]` inserted on next first (x)

`push-back(0)`
`push-front(0)`
`pop-front(0)`
`pop-back(0)`

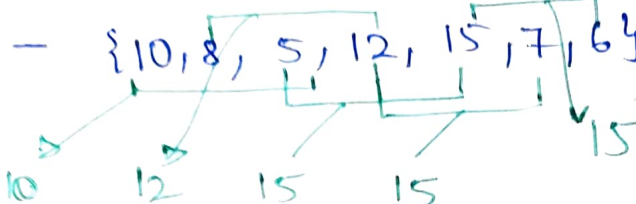
Data structure with min/max operations.

if $\text{insertMin}(x)$ \leftarrow insert at front
 $\text{insertMax}(x)$ \leftarrow insert at end

min \leftarrow \rightarrow max (Deque)
sorted $O(1)$

Maximum of all subarray of size k

I/P - $\{10, 8, 5, 12, 15, 7, 6\}$. $k=3$



no of output
 $= \underline{\underline{n - k + 1}}$

O/P $\rightarrow \{10, 12, 15, 15, 15\}$.

Naive approach $O(n^2)$

find the max of all the windows by using two loops.

Efficient Approach $O(n)$

$\{10, 8, 5, 12, 15, 7, 6\}$.

- we will make a deque of size k, whenever we see an element ~~if~~ greater than the front of deque, we remove the front element and ~~into~~ insert the element
uggghh...

let me show you by an example okay :)

10	8	5
----	---	---

when we see 12 (Remove 10 & all the elements on right)

12		
----	--	--

The idea is whenever, use all a larger element, smaller element is of no use to us.

$i=0$ deg-

10		
----	--	--

 $i=1$ deg-

10	8	
----	---	--

 $i=2$ deg-

10	8	5
----	---	---

 $i=3$ deg-

12		
----	--	--

$i=4$ deg-

15		
----	--	--

 $i=5$ deg-

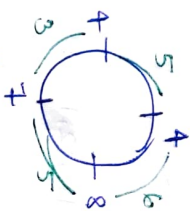
15	7	
----	---	--

 $i=6$ deg-

15	7	6
----	---	---

Circular Tour ~~***~~

I/P $\rightarrow \{4, 8, 7, 4\} \leftarrow$ petrol
 $\{6, 15, 3, 5\} \leftarrow$ dist



We maintain a deque and add the petrol stops till the cur petrol is non-negative. As soon as the cur petrol becomes negative we remove one petrol spot from the front of queue.

petrol - $\{50, 10, 60, 100\}$
 $\{30, 20, 100, 10\}$

deque $\{ \}$ cur-petrol = 0

0

 cur-p = (50-30)+0 = 20

0	1
---	---

 cur-p = 20 + (10-20) = 10

0	1	2
---	---	---

 cur-p = 10 + (60-100) = -30.

1	2
---	---

 cur-p = -30 - (50-30) = -50

2

 cur-p = -50 - (10-20) = -40

$\{ \}$

cur-p = -40 - (100-100) = 0

3

cur-p = 0 + 100 - 0 = 100

3	0
---	---

cur-p = 100 + (50-30) = 120

3	0	1
---	---	---

cur-p = 120 + (10-20) = 110

3	0	1	2
---	---	---	---

cur-p = 110 + (60-100) = 110 - 40 = 70

Ans-3 9 front

If we are traversing for $P_0 \dots P_i$ and at P_i the cur petrol becomes negative, the claim is that none of the points from P_0 to P_i can be a valid solution.

int firstPetrolPump (int petrol[], int dist[], int n)

int start = 0, cur-p = 0;

int prev-p = 0;

for (int i=0; i < n; i++)

cur-p += (petrol[i] - dist[i]);

if (cur-p < 0)

start = i+1;

prev-p += cur-p;

cur-p = 0;

}

return ((cur-p + prev-p) >= 0) ? (start+1) : -1;