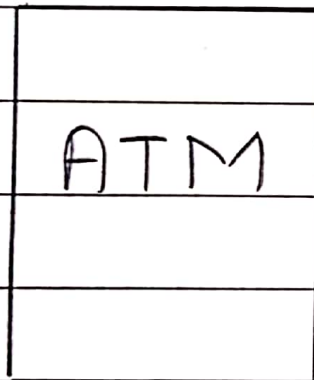


29/04/2023

Queue

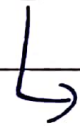
Data structure which follows FIFO i.e first in first out.



P1

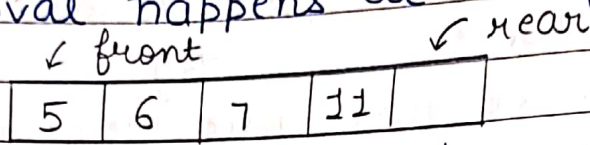
P2

P3

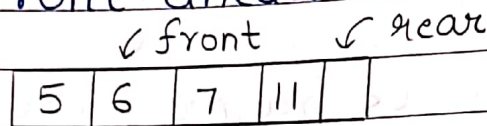


Has come first and hence will go inside the ATM Shop and come first out of the ATM.

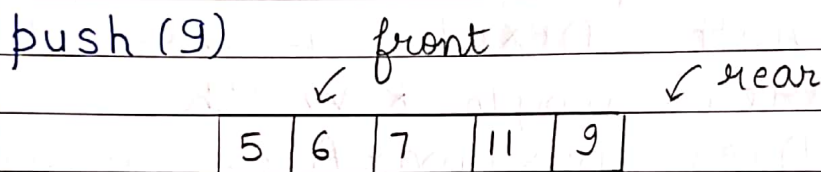
- * Insertion happens at the rear end.
- * Removal happens at the front end.



Now if we want to do removal, simply move front ahead.



Note → Queues are majorly used in graphs, traversals and sliding window approach.



Using STL

We need to include header file i.e
#include <queue>

(i) Creation of queue
queue <int> q;

The above queue will store integer values.

(ii) Inserting element in queue
q.push(5);
q.push(10);
q.push(15);

(iii) Remove element in queue
q.pop();

Here 5 will be removed from the queue.

(iv) Checking empty or not

`q.empty()` \rightarrow True if size = 0 else return False.

(v) Size of queue

`cout << q.size();` \rightarrow 2 will be the size here

(vi) Checking front element of the queue

`cout << q.front();` \rightarrow 10 will be the ans here

Implementation of queue using array

We will be requiring front, rear, array & size.

Initially front = 0 and rear = 0

(i) Push

Check queue is full or not. If not empty, insert.

```
if (rear == size) { //Condition of full
```

```
    cout << "Queue is Full";
```

```
}
```

```
else {
```

```
    arr[rear] = data;
```

```
    rear ++;
```

```
}
```

(ii) Pop

First check if queue is empty or not. If not empty move front ahead.

```
if (front == rear) {
```

```
    cout << "Empty";
```

```
}
```

```
else {
```


3

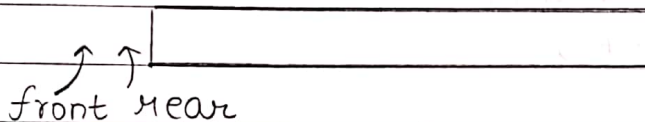
↓ mean

is

rear - front size. Hence $6 - 0 = 6$ is the size.

Circular Queue

Here let's implement with $\text{front} = -1$ and $\text{rear} = -1$ as initial things.



We need to handle insertion of 1st element explicitly when the queue is empty.

(i) Push operation

- ① Full \Rightarrow Then we can't insert and hence simply display message.
- ② First element insert $\Rightarrow \text{front}++$, $\text{rear}++$ & then $\text{arr}[\text{rear}] = \text{data}$;
- ③ Establish circular nature \Rightarrow when $\text{rear} == n-1$ & $\text{front} != 0$, then make $\text{rear} = 0$ and then $\text{arr}[\text{rear}] = \text{data}$
- ④ Normal case $\Rightarrow \text{rear}++$ and $\text{arr}[\text{rear}] = \text{data}$.

```
void push (int data) {
    // Queue is full (One condition pending)
    if ((front == 0 && rear == size-1) ||
        (rear == (front-1) % (size-1))) {
        cout << "cannot insert" ; }

    // Single element case
    else if (front == -1) {
        front = rear = 0;
        arr[rear] = data;
    }
}
```


// Circular nature

```
else if (front != 0 && rear == size-1) {  
    rear = 0;  
}
```

// Normal flow

```
else {
```

```
    arr[rear] = data;
```

```
    rear++;
```

```
}
```

```
}
```

rear front

↓ ↓

2	3	4	1	5
---	---	---	---	---

} One more case in which queue is full.

(ii) Pop operation

① Check for empty

② Single element

③ Circular nature

④ Normal flow

```
void pop() {
```

```
    // Empty check
```

```
    if (front == -1) {
```

```
        cout << "Empty queue";
```

```
    }
```

```
    // Single element
```

```
    else if (front == rear) {
```

```
        arr[front] = -1;
```

```
        front = rear = -1;
```

```
    }
```

```
    // Circular nature
```

```
    else if (front == size-1) {
```

↳ Here we don't have to check rear != 0 as we delete & not insert.

```
front = 0;
```

```
}
```

```
else { // Normal flow
```

```
front ++;
```

```
}
```

```
}
```

i/p restricted queue

insertion from rear end.

removal from both rear end & front end.

o/p restricted queue

insertion from both end i.e front & rear.

removal from front end only

Doubly ended queue (deque)

It is pronounced as deck. We can do push and pop both operations from both ends i.e front and rear end.

(i) pushRear

Same code as that of push operation of the circular queue. Circular condition will depend on the question.

(ii) pushFront

Full condition & single (first) element case will be same.

Circular nature

front == 0 & rear != size - 1, then
make front = size - 1

Normal flow arr[front] = data;
Instead of rear++, simply do front--

(iii) popFront()
Same code as that of pop of the circular queue.

(iv) popRear() single
Empty and element case will be same.

Circular nature

rear == 0 \Rightarrow rear = size - 1;

Normal flow

Simply do rear--

STL for deque

First of all we need to include the header file.

#include <deque>

(i) Creation

deque <int> d;

(ii) push operations in deque

d.push_front(5);

d.push_front(10);

d.push_back(15);

d.push_back(20);

10 5 20 15

(iii) Size of deque

```
cout << "size = " << d.size();
```

(iv) Checking front & rear

```
cout << d.front(); // Front element
```

```
cout << d.back(); // Rear element
```

(v) Checking empty or not

`d.empty()` → True if empty otherwise false.