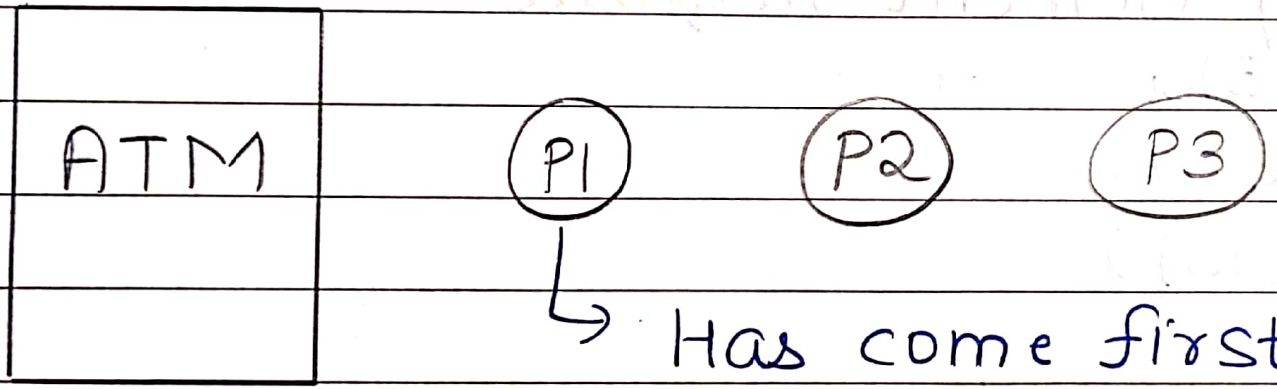
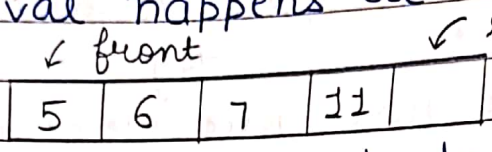
29/04/2023

Queue

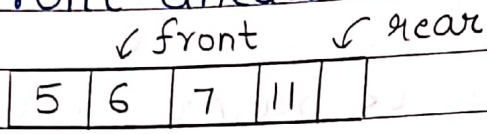
Data structure which follow FIFO i.e first in first out.



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

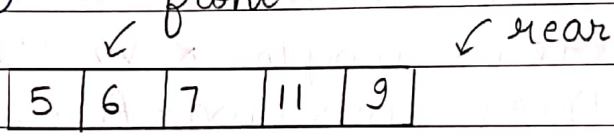


Now if we want to do removal, simply move front ahead.



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

push (9)



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() → True if size = 0 else return False.

(v) Size of queue
cout << q.size(); → 2 will be the size here

(vi) Checking front element of the queue
cout << q.front(); → 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 {
```

```
arr [front] = -1;
front ++;
// Check if queue empty or not. If
empty then make default behaviour
if (front == rear) {
        front = 0;  } Default behaviour
        rear = 0;   }
}
```

To utilize the space.

(iii) getFront ()

Check empty queue or not. If queue is empty, then simply print empty else return arr [front].

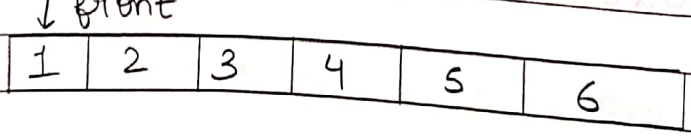
```
if (front == rear) {
        cout << "Empty";
}
else {
    return arr [front];
}
```

(iv) isEmpty ()

front == rear ⇒ Empty queue & hence return true. If not empty then return false.

(v) getSize ()

Simply return rear - front here. Here +1 is not done as rear is pointing to the empty location.

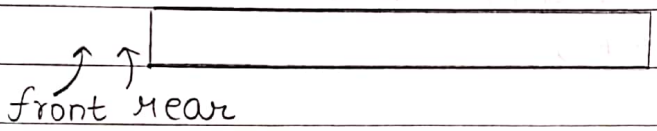


is

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

Circular Queue

Here let's implement with front = -1 and rear = -1 as initial things.



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

(i) Push operation

① Full ⇒ Then we can't insert and hence simply display message.

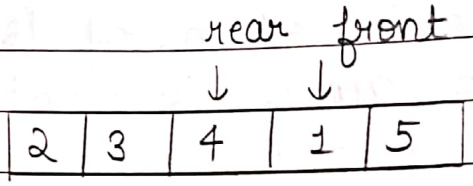
② First element insert ⇒ front ++ , rear ++ & then arr [rear] = data;

③ Establish circular nature ⇒ when rear == n-1 && front != 0 , then make rear = 0 and then arr [rear] = data

④ Normal case ⇒ rear ++ and arr [rear] = 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 ++;
}
}
```



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 element case will be Empty and same.

Circular nature

rear == 0 ⇒ 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.