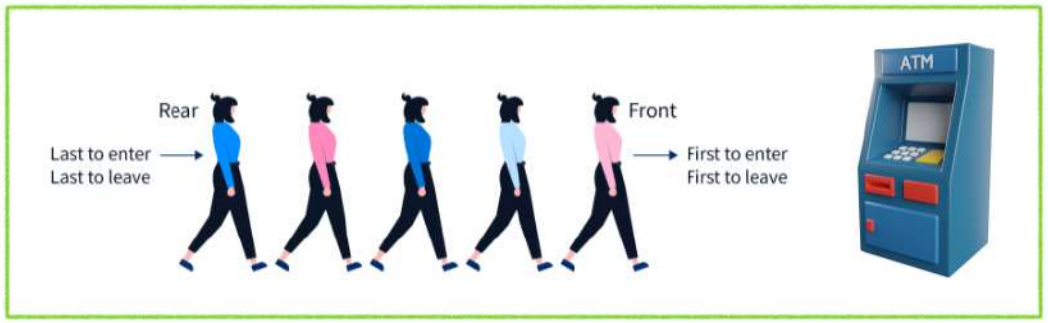
21/11/2023

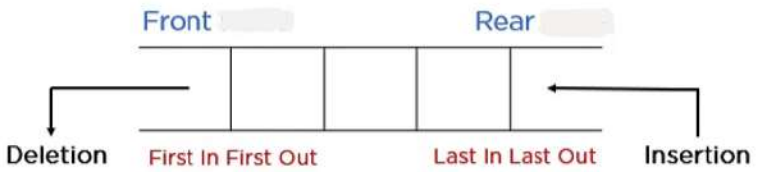
QUEUE CLASS - 1

1. What is queue?

*Queue is linear data structure which operates in a **FIFO** (First in First out) principle.*
Queue has two pointers are front and rear.

Note:
*All insertions are made at one end, called **REAR**.*
*All deletions are made at one other end, called **FRONT**.*



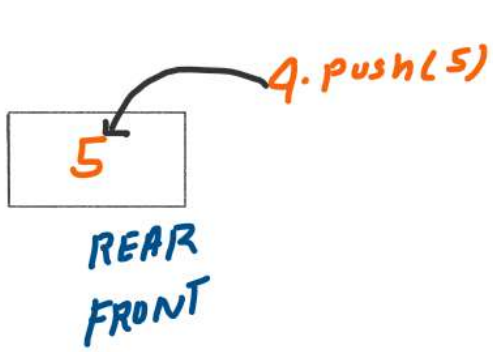


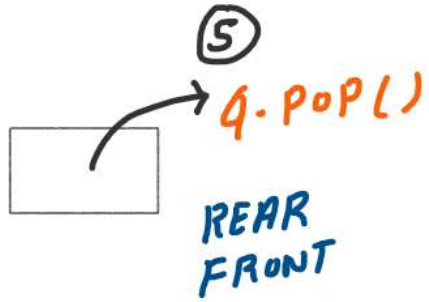
Queue Representation

```cpp
// STL Queue
#include<iostream>
#include<queue>
using namespace std;

int main(){
    // Create Queue
    queue<int> q;

    // Insertion
    q.push(5);

    // Size
    cout<< "Size of Queue: " << q.size() << endl;

    // Empty
    if(q.empty()){
        cout<< "Queue is empty" << endl;
    }
    else{
        cout<< "Queue is not empty" << endl;
    }

    // Deletion
    q.pop();

    q.push(10);
    q.push(20);
    q.push(30);

    // Front
    cout<< "Front element of queue: " << q.front() << endl;

    // Rear
    cout<< "Rear element of queue: " << q.back() << endl;

    return 0;
}
```

OUTPUT:

```
Size of Queue: 1
Queue is not empty
Front element of queue: 10
Rear element of queue: 30
```



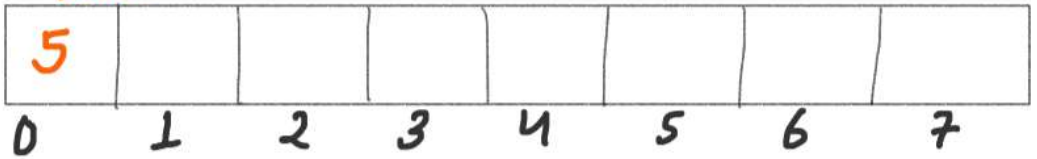


2. Implementation of Queue using a Dynamic Array

```
Queue {
    array
    size
    front
    rear
    ===
}
```

XFRONT=-1
XREAR=-1

Front=0
REAR=0




```
q.push(5) {
```

UNDERFLOW

Empty

Isse bhool jaya →

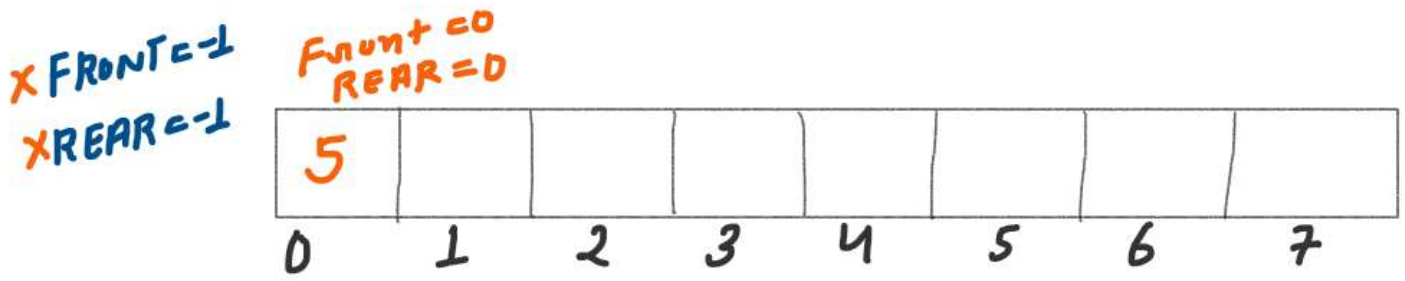
```
front++
rear++
arr[rear]=data
}
```

NORMAL

```
rear++
arr[rear]=data
```

Overflow

```
rear==size-1
```



4. pop() {

UNDERFLOW
EMPTY

first == -1 &&
rear == -1

Isse Bhool Jaiyo Nahi

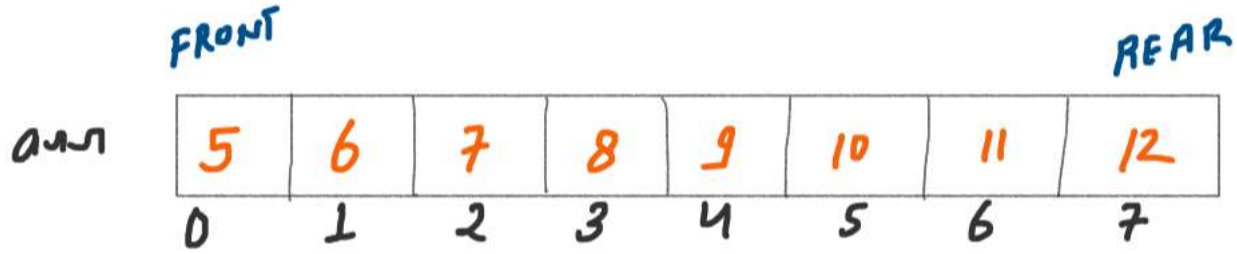
SINGLE ELEMENT

first == rear
arr[front] = -1;
front = -1
rear = -1

NORMal

arr[front] = -1
front++

}



q.getSize() {

empty

return 0

Isse mil bhool jata hu

NORMAL

size = Rear - front + 1;

}

q.empty() {

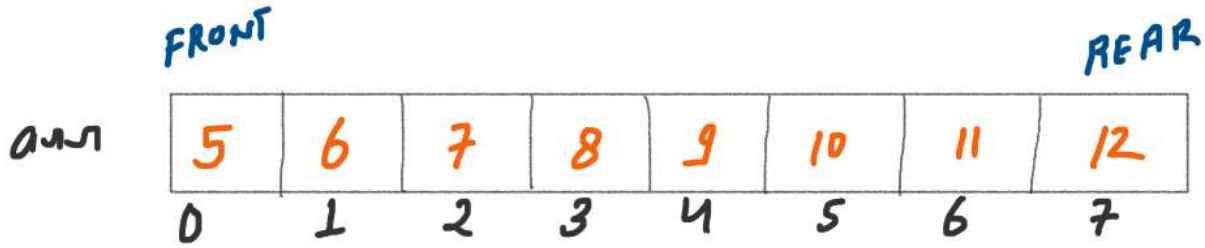
if(Front == -1 && rear == -1)

↳ Return True

else

↳ Return false

}



q. front () {

Empty

if(front == -1)

↳ Return -1

}

NORMal

return arr[front]

q. rear () {

Empty

if (rear == -1)

↳ return -1

NORMal

return arr[rear]

```cpp
// 2. Implementation of Queue using a Dynamic Array
#include<iostream>
using namespace std;

class Queue
{
    public:
        int* arr;
        int size;
        int front;
        int rear;

        Queue(int size){
            arr = new int[size];
            this->size = size;
            front = -1;
            rear = -1;
        }

    1   // Insertion
        void push(int val){...}

    2   // Deletion
        void pop(){...}

    3   // Size of Queue
        int getSize(){...}

    4   // Queue is empty or not
        bool isEmpty(){...}

    5   // Front element of queue
        int getFront(){...}

    6    // Rear element of queue
        int getRear(){...}

        // Optional method just for testing purpose
        void print(){
            cout<< "Front Index: "<<front<<" | Rear Index: "<<rear<<endl;
            cout<< "Size of queue: "<< getSize() <<endl;
            cout<< "Printing Queue: ";
            for(int i=0; i<size; i++){
                cout<< arr[i] << " ";
            }
            cout<<endl<<endl;
        }

};
```

1

```cpp
// Insertion
void push(int val){
    // Overflow Queue
    if(rear == size - 1){
        cout<< "Overflow Queue" << endl;
        return;
    }
    // Empty (Isse me bhool jata hu)
    else if(front == -1 && rear == -1){
        front++;
        rear++;
        arr[rear] = val;
    }
    // Normal
    else{
        rear++;
        arr[rear] = val;
    }
}
```

2

```cpp
// Deletion
void pop(){
    // Underflow Queue
    if(front == -1 && rear == -1){
        cout<< "Underflow Queue" << endl;
        return;
    }
    // Single Element Queue (Isse me bhool jata hu)
    else if(front == rear){
        arr[front] = -1;
        front = -1;
        rear = -1;
    }
    // Normal
    else{
        arr[front] = -1;
        front++;
    }
}
```

3

```
// Size of Queue
int getSize(){
    // Empty Queue (Isse me bhool jata hu)
    if(front == -1 && rear == -1){
        return 0;
    }
    // Normal
    else{
        return (rear - front + 1);
    }
}
```

4

```
// Queue is empty or not
bool isEmpty(){
    // Empty Queue
    if(front == -1 && rear == -1){
        return true;
    }
    else{
        return false;
    }
}
```

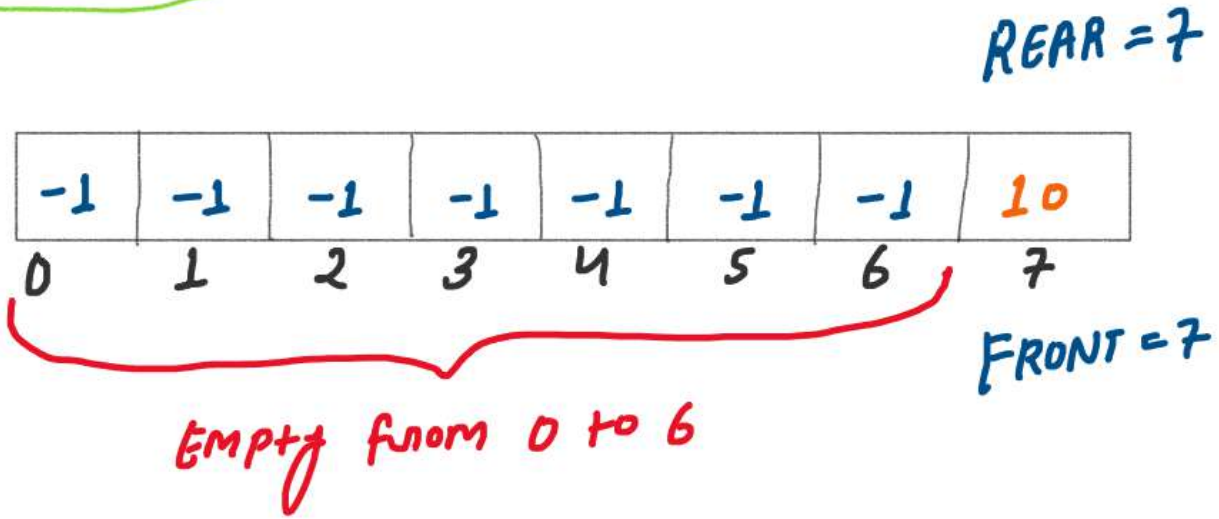
5

```
// Front element of queue
int getFront(){
    // Empty Queue
    if(front == -1){
        return -1;
    }
    // Noraml
    else{
        return arr[front];
    }
}
```

6

```
// Rear element of queue
int getRear(){
    // Empty Queue
    if(rear == -1){
        return -1;
    }
    // Noraml
    else{
        return arr[rear];
    }
}
```

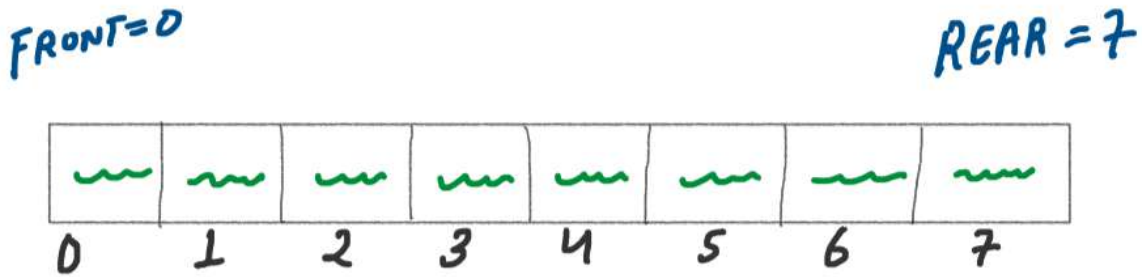
Disadvantage of above code



Q. push(800)

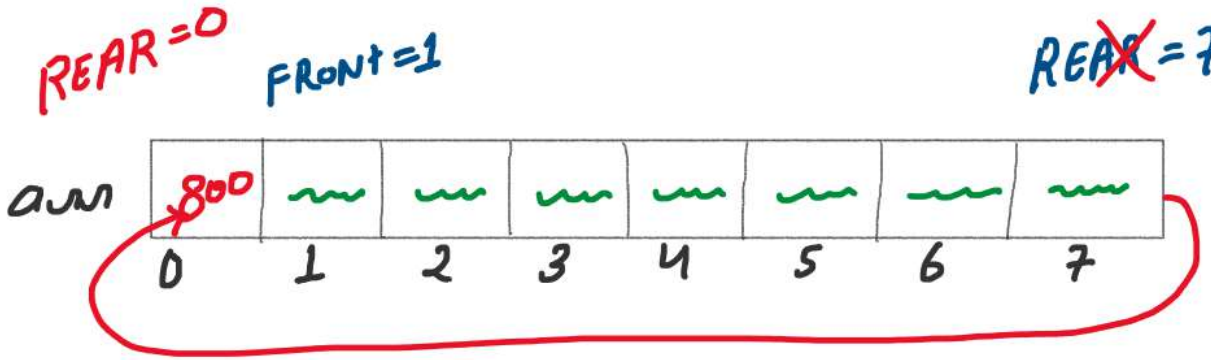
> ⭐ *JAB front and rear pointer array ke last index par honge*
> *To iss code ki help se hum 800 ko insert nhi kar skte hai*
> *joki queue ki algorithm ke according array ki 0th index par insert hona chaiye*
> ***iska mtlb memory space waste ho rha hai --> Iska solution Circular Queue hai***

3. Implementation of Circular Queue using a Dynamic Array



1. full queue if (front == 0 && rear == size-1)
2. Empty queue if (front == -1 && rear == -1)
3. Single Element queue if (front == rear)


```
CQueue {
    arr
    size
    front
    rear
}
```



push()

Overflow

```
if(front==0 &&
   rear==size-1)
  ↳ return
```

Empty

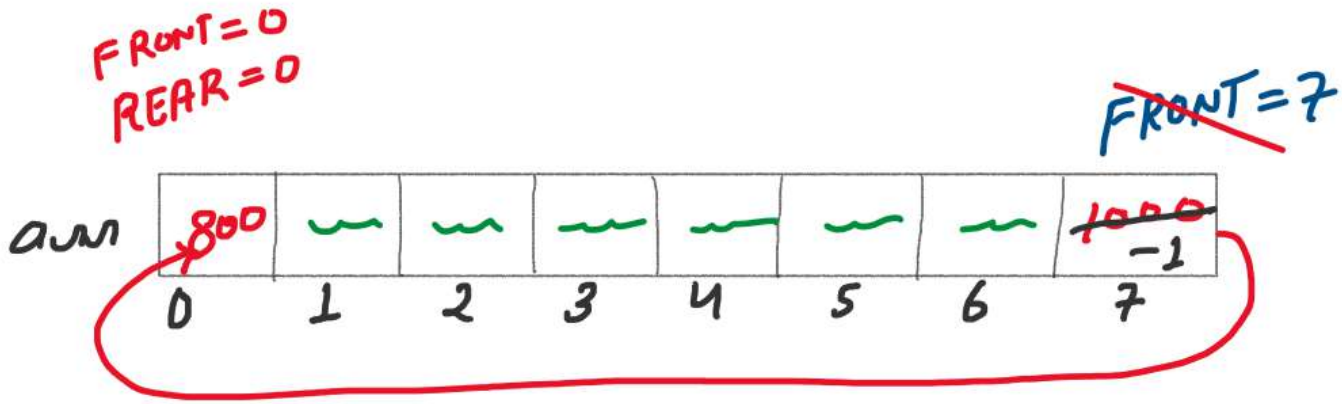
```
if(front==-1 &&
   rear==-1)
  ↳ front++;
    rear++;
    arr[rear]=val;
```

Circular

```
if(rear==size-1
   && front!=0)
  ↳ rear=0;
    arr[rear]=val;
```

Normal

```
rear++;
arr[rear]=val;
```



POP()

Underflow

if(front == -1 || rear == -1)

↳ Empty Queue

Single Element

if(rear == front)

↳ arr[front] = -1

front = -1;

rear = -1;

Circular

if(front == size-1)

↳ arr[front] = -1

front = 0;

Normal

arr[front] = -1;

front ++;

```cpp
// 3. Implementation of Circular Queue using a Dynamic Array
#include<iostream>
using namespace std;

class CQueue
{
    public:
        int* arr;
        int size;
        int front;
        int rear;

        CQueue(int size){
            arr = new int[size];
            this->size = size;
            front = -1;
            rear = -1;
        }

    1   // Insertion
        void push(int val){...}

    2   // Deletion
        void pop(){...}

        // Optional method just for testing purpose
        void print(){
            cout<< "Front Index: "<<front<<" | Rear Index: "<<rear<<endl;
            cout<< "Printing Queue: ";
            for(int i=0; i<size; i++){
                cout<< arr[i] << " ";
            }
            cout<<endl<<endl;
        }
};
```

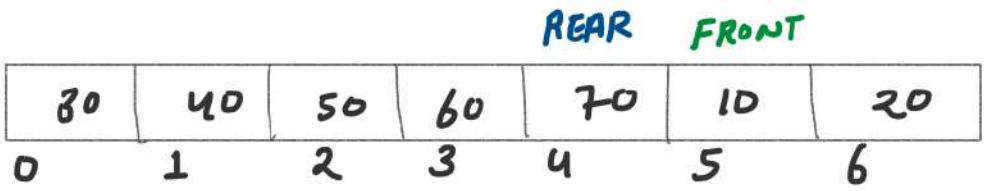
1

```
// Insertion
void push(int val){
    // Overflow
    if(front == 0 && rear == size-1){
        cout<< "Overflow Queue" << endl;
        return;
    }
    // Empty queue
    else if(front == -1 && rear == -1){
        front++;
        rear++;
        arr[rear] = val;
    }
    // Circular queue
    else if(rear == size-1 && front != 0){
        rear = 0;
        arr[rear] = val;
    }
    // Normal
    else{
        rear++;
        arr[rear] = val;
    }
}
```

2

```
// Deletion
void pop(){
    // Underflow
    if(front == -1 && rear == -1){
        cout<< "Underflow Queue" << endl;
        return;
    }
    // Single element queue
    else if(front==rear){
        arr[front] = -1;
        front = -1;
        rear = -1;
    }
    // Circular queue
     else if(front == size-1){
        arr[front] = -1;
        front = 0;
    }
    // Normal
    else{
        arr[front] = -1;
        front++;
    }
}
```

Above Code Me Abhi Bhi EK Problem Hai --> OVERFLOW CONDITION Ki



Isse me bhool jata HU

```
if( REAR == FRONT - 1){
    cout<< "Overflow" << endl;
}
```

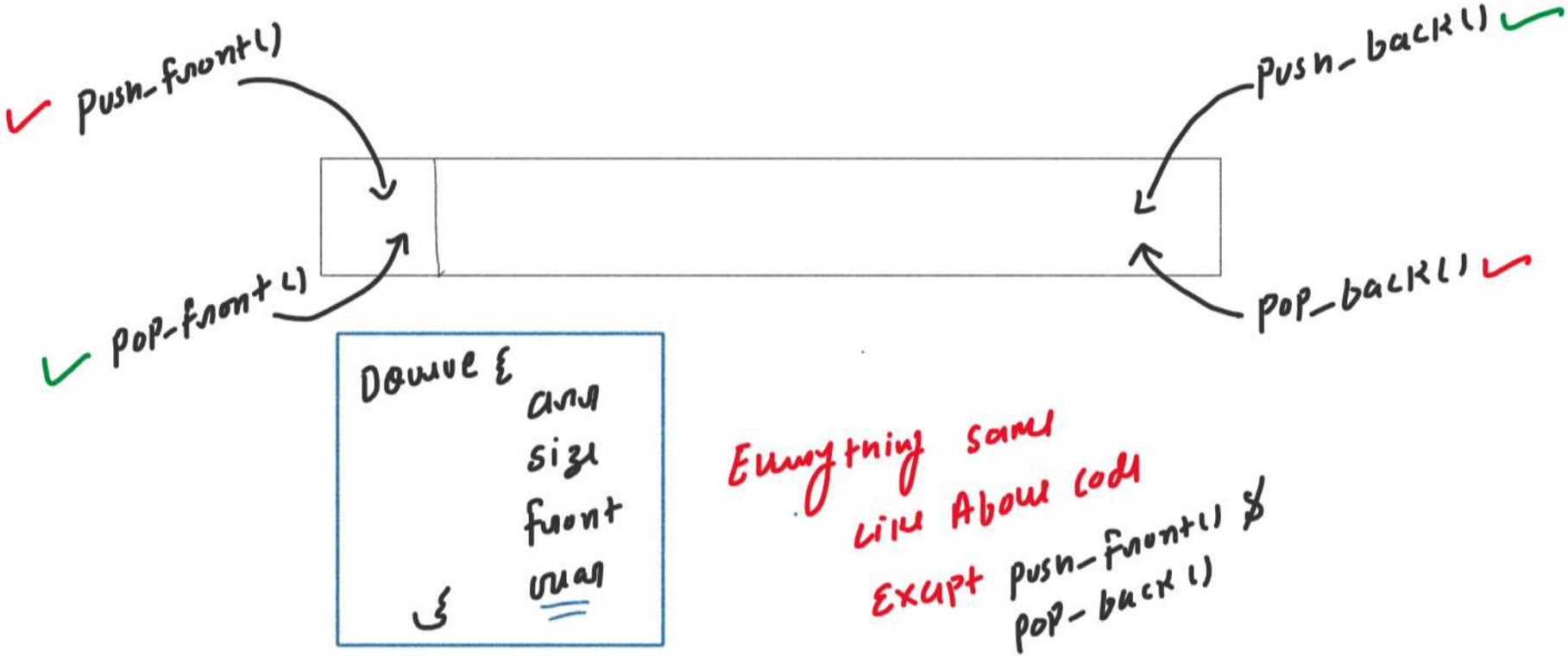
q.push → 10, 20, 30, 40, 50, 60, 70, 80 → Overflow Ho jayega

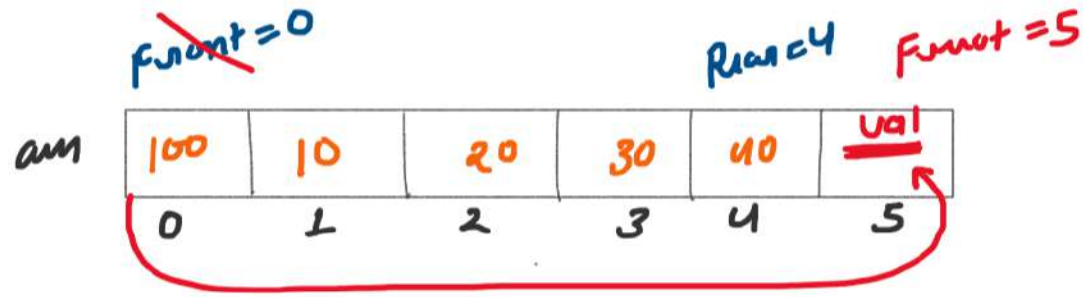
1

```cpp
// Insertion
void push(int val){
    // Overflow
     if( (front == 0 && rear == size-1) || (rear == front-1) ){
        cout<< "Overflow Queue" << endl;
        return;
    }
    // Empty queue
    else if(front == -1 && rear == -1){
        front++;
        rear++;
        arr[rear] = val;
    }
    // Circular queue
    else if(rear == size-1 && front != 0){
        rear = 0;
        arr[rear] = val;
    }
    // Normal
    else{
        rear++;
        arr[rear] = val;
    }
}
```

AB CODE PERFECT HO CHUKA HAI

4. Implementation of Circular Double Ended Queue using a Dynamic Array





Push_Front()

Overflow

```
if((front==0 &&
   rear==size-1) ||
   (rear==front-1))
```

↳ Full Hai Queue

Empty

```
if(Front==-1 &&
   Rear==-1)
```

↳ Front++
Rear++
arr[Front] = val;

Circular

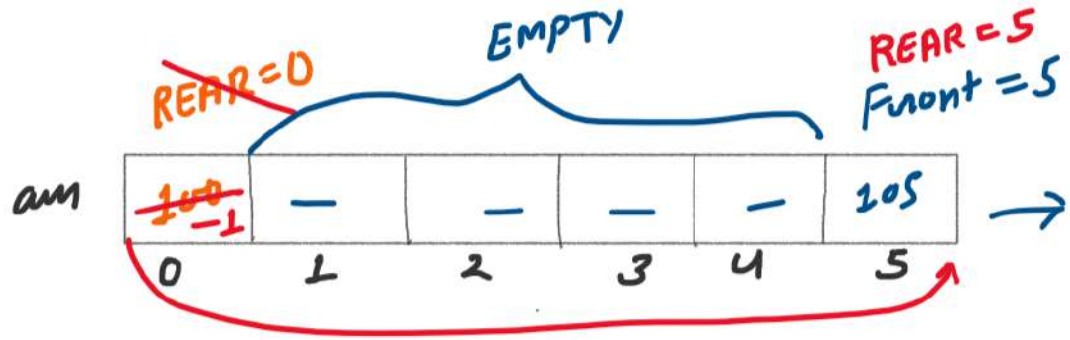
```
if(front==0 &&
   rear!=size-1)
```

↳ front = size-1;
arr[front] = val;

Normal

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

pop_back()



Underflow

if(front == -1 &&
rear == -1)

↳ Empty queue Hai

Single element

if(rear == front)

↳ arr[rear] = -1 ;

rear = -1 ;

front = -1 ;

Circular

if(rear == 0)

↳ arr[rear] = -1

rear = size-1 ;

Normal

arr[rear] = -1;

rear -- ;

```cpp
// 4. Implementation of Circular Double Ended Queue using a Dynamic Array
#include<iostream>
using namespace std;

class CDQueue
{
    public:
        int* arr;
        int size;
        int front;
        int rear;

        CDQueue(int size){
            arr = new int[size];
            this->size = size;
            front = -1;
            rear = -1;
        }

        // insertion and deletion from front
    1   void pushFront(int val){...}

    2   void popFront(){...}

        // insertion and deletion from rear
    3   void pushBack(int val){...}

    4   void popBack(){...}

        // Optional method just for testing purpose
        void print(){
            cout<< "Front Index: "<<front<<" | Rear Index: "<<rear<<endl;
            cout<< "Printing Queue: ";
            for(int i=0; i<size; i++){
                cout<< arr[i] << " ";
            }
            cout<<endl<<endl;
        }
};
```

1

```
// insertion and deletion from front
void pushFront(int val){
    // Overflow
     if((front == 0 && rear == size-1) || (rear == front-1)){
        cout<< "Overflow Queue" << endl;
        return;
    }
    // Empty queue
    else if(front == -1 && rear == -1){
        front++;
        rear++;
        arr[front] = val;
    }
    // ⭐Circular queue
    else if(front == 0 && rear != size - 1){
        front = size - 1;
        arr[front] = val;
    }
    // Normal
    else{
        front--;
        arr[front] = val;
    }
}
```

2

```
void popFront(){
    // Underflow
    if(front == -1 && rear == -1){
        cout<< "Underflow Queue" << endl;
        return;
    }
    // Single element queue
    else if(front==rear){
        arr[front] = -1;
        front = -1;
        rear = -1;
    }
    // Circular queue
     else if(front == size-1){
        arr[front] = -1;
        front = 0;
    }
    // Normal
    else{
        arr[front] = -1;
        front++;
    }
}
```

3

```
// insertion and deletion from rear
void pushBack(int val){
    // Overflow
     if((front == 0 && rear == size-1) || (rear == front-1)){
        cout<< "Overflow Queue" << endl;
        return;
    }
    // Empty queue
    else if(front == -1 && rear == -1){
        front++;
        rear++;
        arr[rear] = val;
    }
    // Circular queue
    else if(rear == size-1 && front != 0){
        rear = 0;
        arr[rear] = val;
    }
    // Normal
    else{
        rear++;
        arr[rear] = val;
    }
}
```

4

```
void popBack(){
    // Underflow
    if(front == -1 && rear == -1){
        cout<< "Underflow Queue" << endl;
        return;
    }
    // Single element queue
    else if(front==rear){
        arr[rear] = -1;
        front = -1;
        rear = -1;
    }
    // ⭐Circular queue
     else if(rear == 0){
        arr[rear] = -1;
        rear = size - 1;
    }
    // Normal
    else{
        arr[rear] = -1;
        rear--;
    }
}
```

Reference:
https://en.cppreference.com/w/cpp/container/queue
https://cplusplus.com/reference/queue/queue/