Data Structures (Lab-6)

Queues

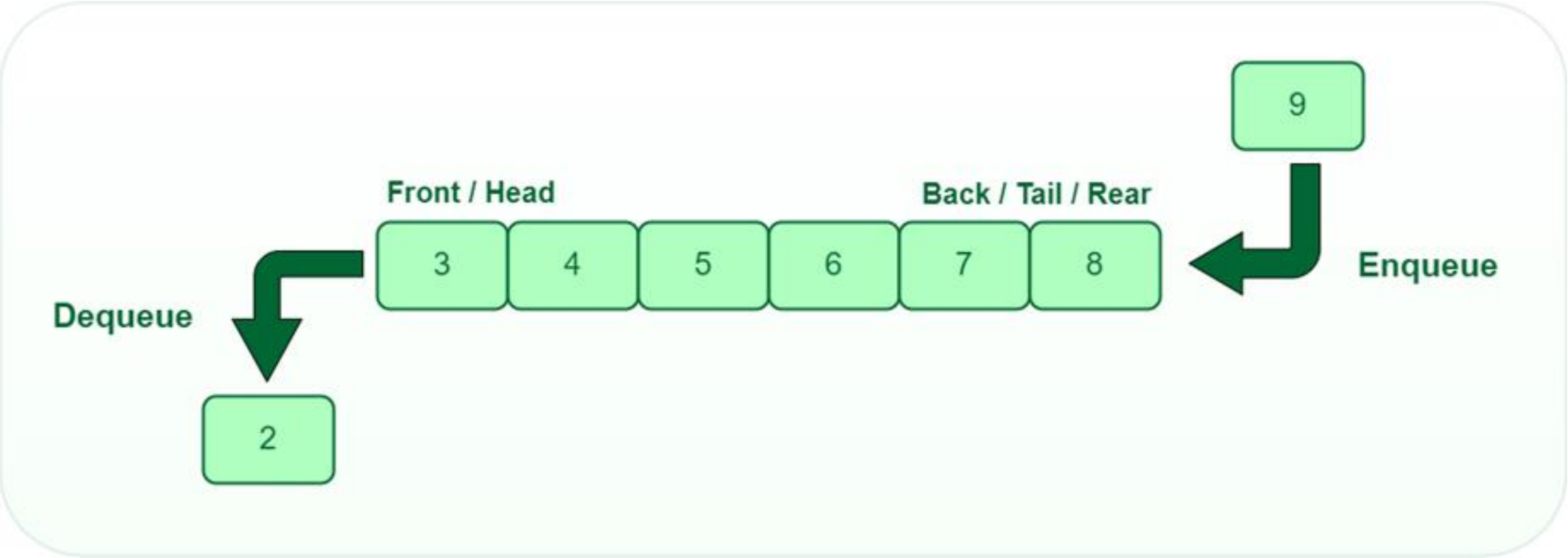
Agenda for Today

- What is Queue
- Queue Applications
- Queue Operations
- Circular Queue
- Queue Implementation using Arrays
- Task

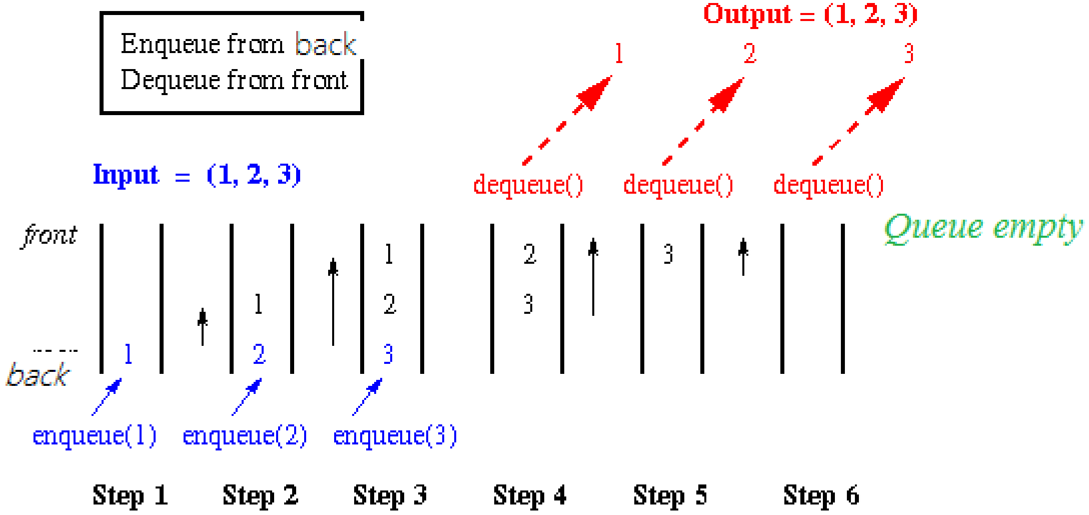
Visual Queue: https://www.cs.usfca.edu/~galles/visualization/QueueArray.html

FIFO
LILO

Queue



Queue in Action



Applications of Queue

- CPU Scheduling and Task Management
- Printers and Resource Sharing
- Call Center Management and Customer Support

Queue Operations

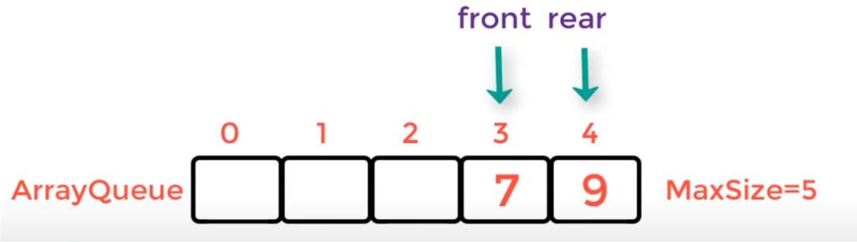
- enqueue()
- dequeue()
- isEmpty()
- isFull()
- front()
- rear()
- queueSize()
- display()
- etc.

Queue Implementation Problem with Arrays

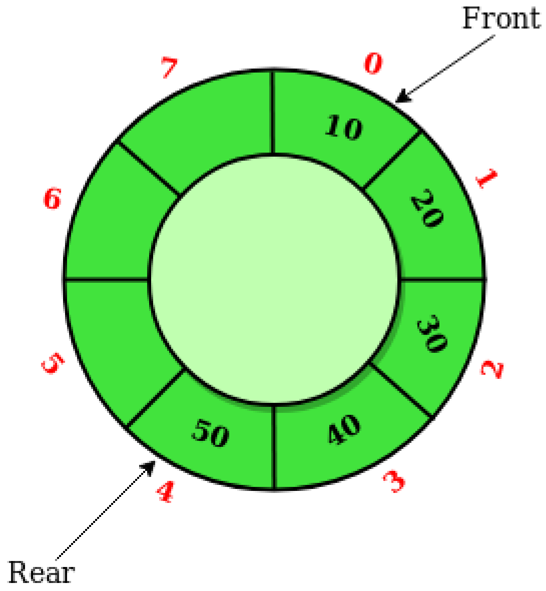
After a number of *enqueue()* & *dequeue()* operations, the **rear/end/tail** will always point to the end of the queue, indicating that the queue is full.

To solve this problem we will use the concept of **Circular Queue**

- Another problem is: fixed size of the queue.



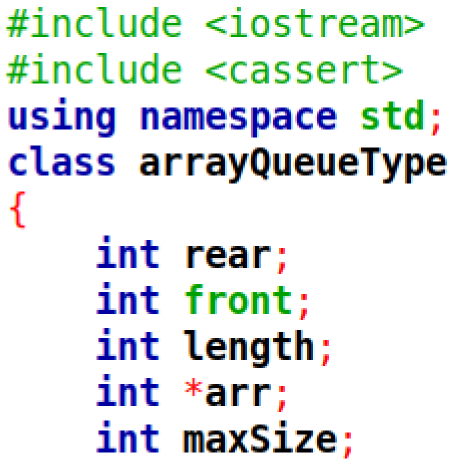
Circular Queue



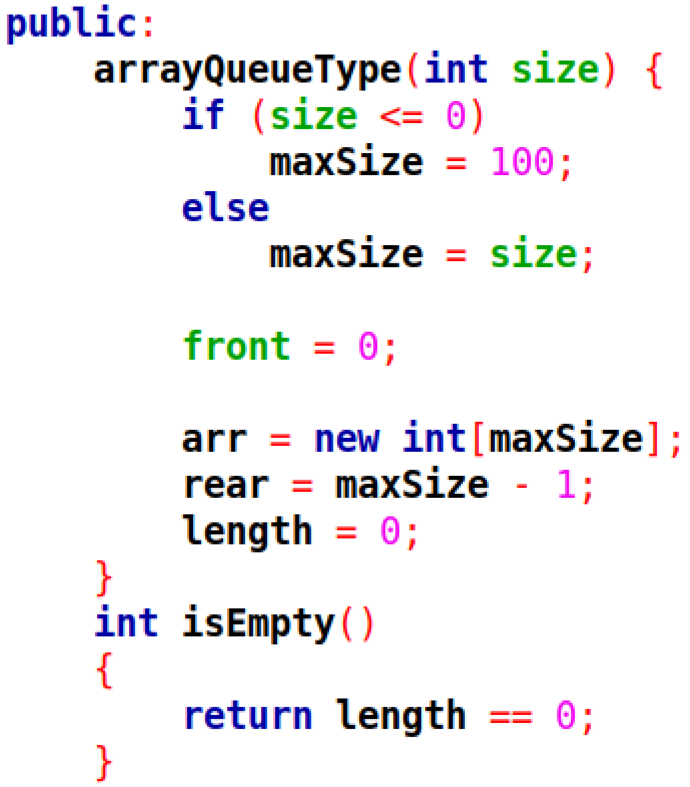
Now, look at the Circular Queue implementation using Arrays. Check circulrQueueArray.cpp.

Let’s implement

- First we need to create class queue that has members:
- Size : int the max size you need
- Front: int the first element in the queue
- Rear: int the last element in the queue
- List : array of int values(or any data type we need)
- Parameterized constructor that take size parameter
- Set size = the size you got from parameter
- Set Front= 0 that refer to the first position will add elemets to it
- Set Rear= maxSize – 1 that refer to the last position can add elemets to it
- Create list it's size that you got from parameter

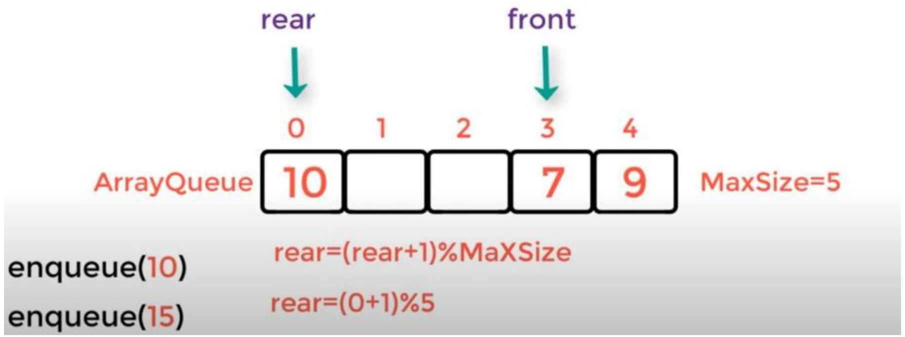


```
#include <iostream>
#include <cassert>
using namespace std;
class arrayQueueType
{
    int rear;
    int front;
    int length;
    int *arr;
    int maxSize;
```



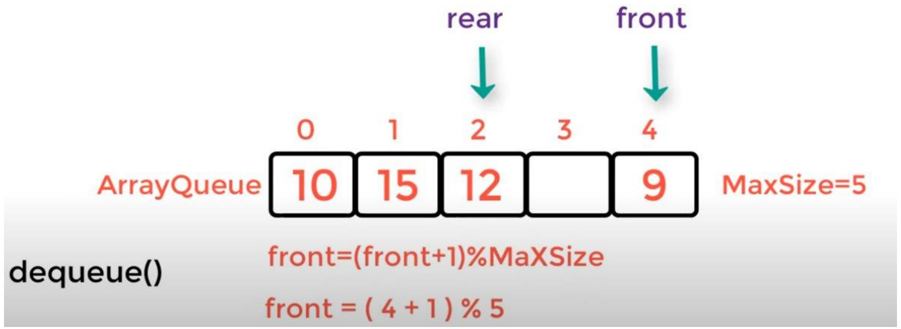
```
public:
    arrayQueueType(int size) {
        if (size <= 0)
            maxSize = 100;
        else
            maxSize = size;

        front = 0;

        arr = new int[maxSize];
        rear = maxSize - 1;
        length = 0;
    }
    int isEmpty()
    {
        return length == 0;
    }
```

Circular Queue (Enqueue)



```
void addQueue(int Element)
{
    if (isFull())
    {
        cout << "Queue Full Can't Enqueue ...!";
    }
    else
    {
        rear = (rear + 1) % maxSize;// as it's circular queue
        arr[rear] = Element;
        length++;
    }


}
```

Circular Queue (Dequeue)



front=(front+1)%MaXSize

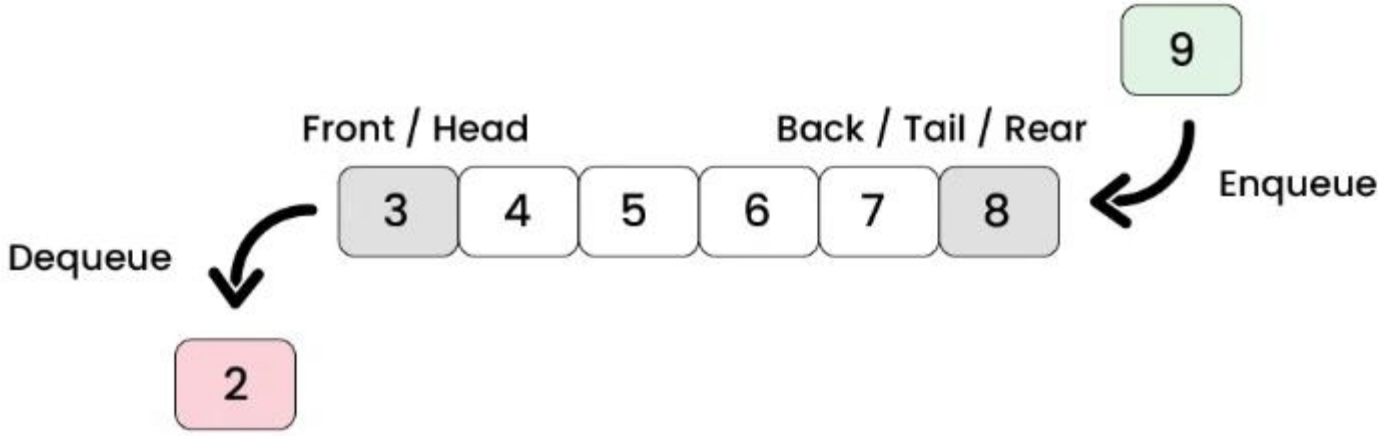
front = (4 + 1) % 5

```
void deleteQueue()
{
    if (isEmpty())
    {
        cout << "Empty Queue Can't Dequeue ...!";
    }
    else
    {
        front = (front + 1) % maxSize;
        --length;

    }


}
```

Drawbacks of implementing queue using Array

- Costly Resizing Operations
- Inefficient Dequeue (Front Removal) Operation
- Limited Capacity for Large Data

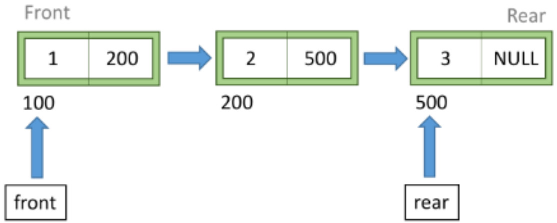
Queue using Linkedl list



Queue using Linkedl list

Each item(node) in queue contains :-

- A piece of data (any type)
- Pointer to the next node in the Queue



Now, look at the Queue using linked list implementation using Arrays. Check QueueLinkedlist.cpp.

Let’s implement

- First we need to create class queue that has members:
- length: int the number of elemets in queue
- Front: int the first element in the queue
- Rear: int the last element in the queue

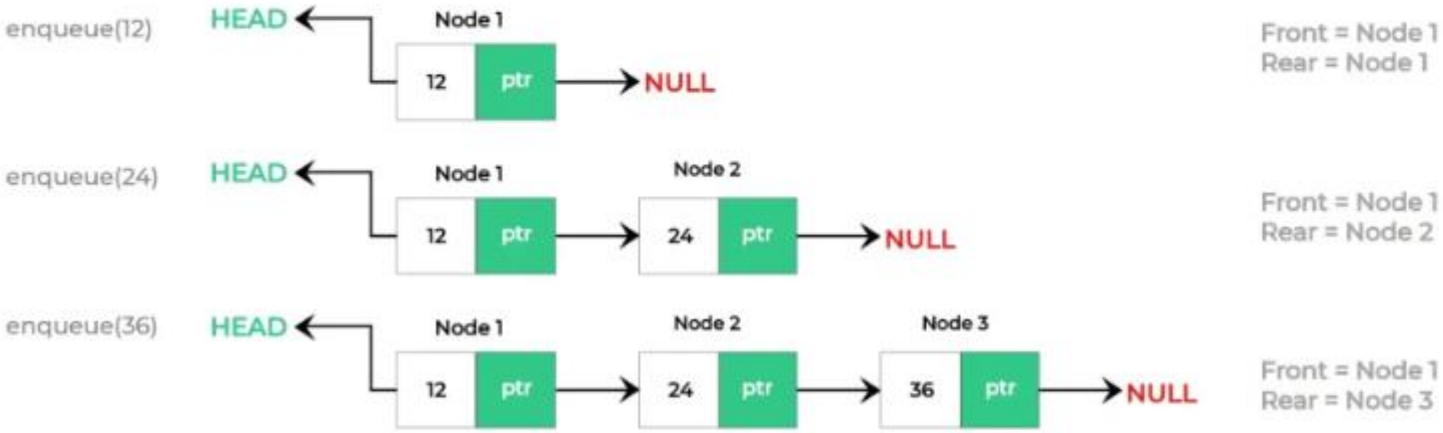
Default constructor

- Set length= 0 as there is no ements in queue
- Set Front,Rear= NULL

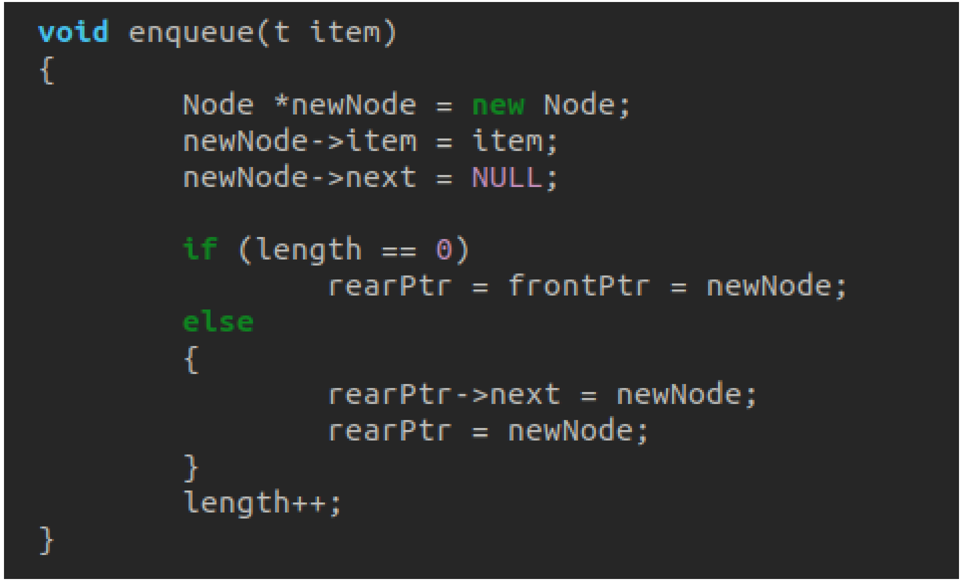
Queue (Enqueue)

- We add elements to the queue from the rear only

Adding the elements into Queue



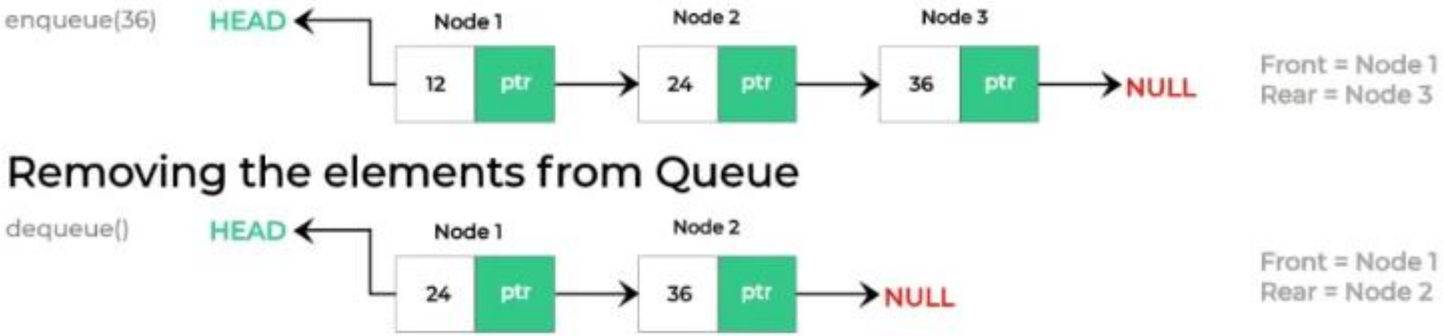
- Check first if there is no nodes in queue so the first node will be front&rear.
- If not next of rear will point to new node Rear will be the new point
- Increament as there new node added



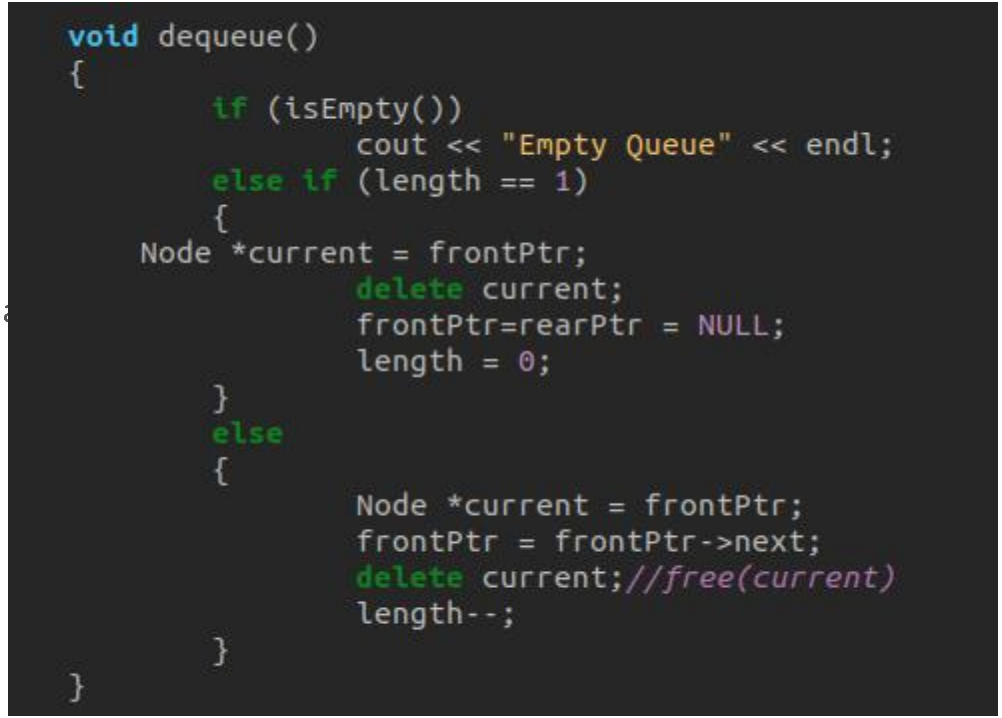
```
void enqueue(t item)
{
        Node *newNode = new Node;
        newNode->item = item;
        newNode->next = NULL;

        if (length == 0)
                rearPtr = frontPtr = newNode;
        else
        {
                rearPtr->next = newNode;
                rearPtr = newNode;
        }
        length++;
}
```

Queue (Dequeue)

- We remove elements from the front only



- Check first if there is no nodes in queue
 so deletion cannot be completed .
- if there is one one so we deleted it and ma
front&rear=NULL.
- If not we remove the front and make front next is the front
- Decrement as there node deleted



```
void dequeue()
{
        if (isEmpty())
                cout << "Empty Queue" << endl;
        else if (length == 1)
        {
    Node *current = frontPtr;
                delete current;
                frontPtr=rearPtr = NULL;
                length = 0;
        }
        else
        {
                Node *current = frontPtr;
                frontPtr = frontPtr->next;
                delete current;//free(current)
                length--;
        }
}
```

Task

- You have to implement a method to check if the queue elements are sorted in ascending order or not, if yes return true, else return false

- **Method Signature**
  - bool sortedAscQueue()

- **Test Case**
  - Queue q = [2, 3, 8, 12, 18, 33]
  - Result => true

Thank you.