QUEUES-II (POINTER BASED)

NATIONAL UNIVERSITY OF TECHNOLOGY (NUTECH)

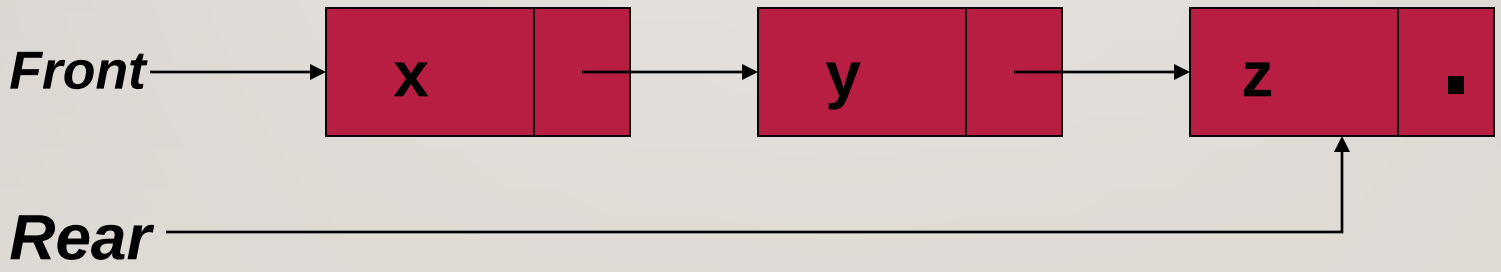
DR. SAMAN RIAZ

LECTURE # 11

A POINTER IMPLEMENTATION OF QUEUES

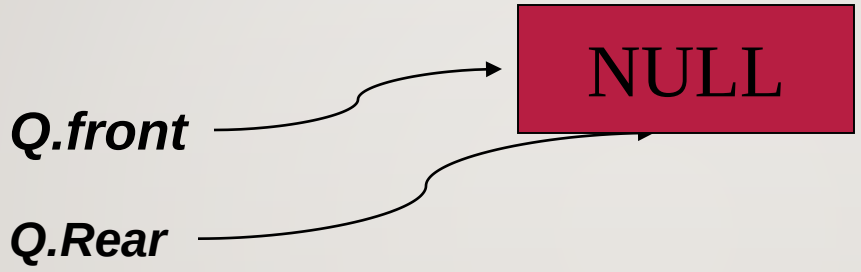
Keep two pointers:

- FRONT: A pointer to the first element of the queue.
- REAR: A pointer to the last element of the queue.



A POINTER IMPLEMENTATION OF QUEUES

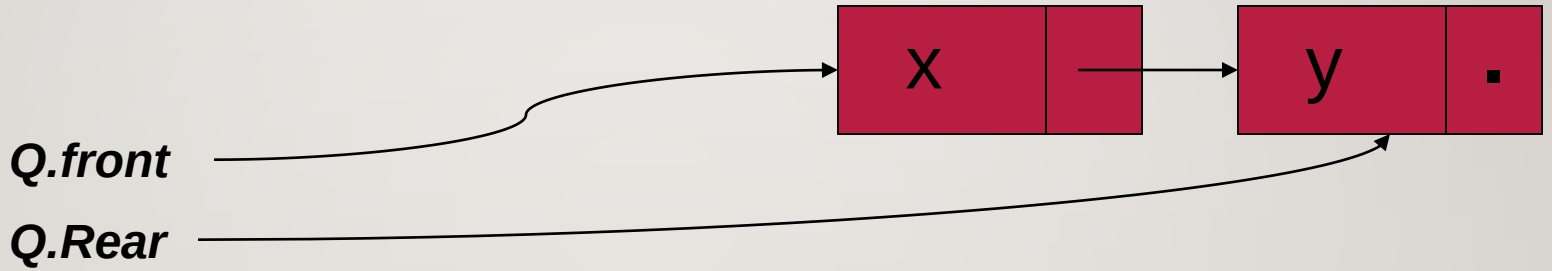
MAKENULL(Q)



ENQUEUE(x,Q)

A POINTER IMPLEMENTATION OF QUEUES

ENQUEUE(y,Q)



DEQUEUE(Q)

A class for Dynamic Queue implementation


```
class DynIntQueue
{
private:
    struct QueueNode
    {
        int value;
        QueueNode *next;
    };

    QueueNode *front;
    QueueNode *rear;
    int numItems;

public:
    DynIntQueue(void);
    ~DynIntQueue(void);
    void enqueue(int);
    int dequeue(void);
    bool isEmpty(void);
    void makeNull(void);
};
```

Implemenaton


```
//************************
// Constructor           *
//************************

DynIntQueue::DynIntQueue(void)
{
    front = NULL;
    rear = NULL;
    numItems = 0;
}

//************************
// Destructor            *
//************************

DynIntQueue::~DynIntQueue(void)
{
    makeNull();
}
```

```
//*****************************************
// Function enqueue inserts the value in num *
// at the rear of the queue.                 *
//*****************************************

void DynIntQueue::enqueue(int num)
{
    QueueNode *newNode;

    newNode = new QueueNode;
    newNode->value = num;
    newNode->next = NULL;
    if (isEmpty())
    {
        front = newNode;
        rear = newNode;
    }
    else
    {
        rear->next = newNode;
        rear = newNode;
    }
    numItems++;
}
```

```
//*********************************************
//   Function dequeue removes the value at the *
// front of the queue, and copies it into num. *
//*********************************************

int DynIntQueue::dequeue(void)
{
    QueueNode *temp;
    int num;
    if (isEmpty())
        cout << "The queue is empty.\n";
    else
    {
        num = front->value;
        temp = front->next;
        delete front;
        front = temp;
        numItems--;
    }
    return num;

}
```

```
//****************************************
// Function isEmpty returns true if the queue *
// is empty, and false otherwise.              *
//****************************************

bool DynIntQueue::isEmpty(void)
{
   if (numItems)
      return false;
   else
      return true;
}
```

```
//*****************************************
// Function makeNull dequeues all the elements  *
// in the queue.                                *
//*****************************************

void DynIntQueue::makeNull(void)
{
   while(!isEmpty())
      dequeue();
}
```

Program


```
// This program demonstrates the DynIntQeue class
void main(void)
{
    DynIntQueue iQueue;

    cout << "Enqueuing 5 items...\n";
    // Enqueue 5 items.
    for (int x = 0; x < 5; x++)
        iQueue.enqueue(x);

    // Deqeue and retrieve all items in the queue
    cout << "The values in the queue were:\n";
    while (!iQueue.isEmpty())
    {
        int value;

         value= iQueue.dequeue();
        cout << value << endl;
    }
}
```

Program Ouput


```
Enqueuing 5 items...
The values in the queue were:
0
1
2
3
4
```