

# Data Structures

---

## **13. Queues Using Linked List**

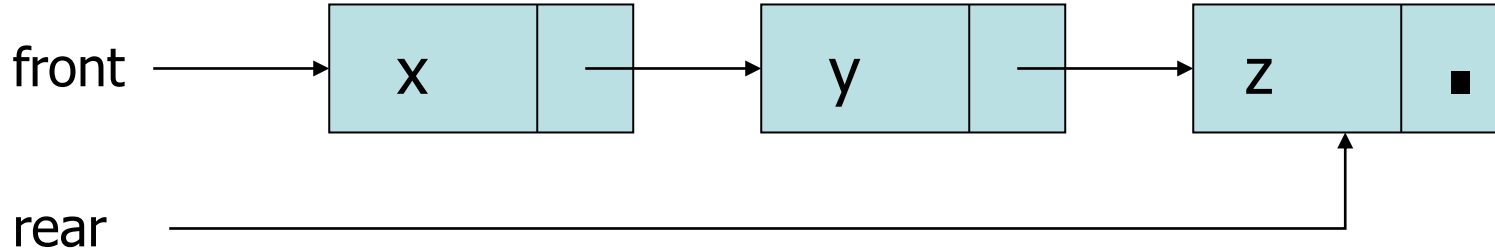
---

# Pointer-based Implementation

# Pointer-Based Implementation of Queues

---

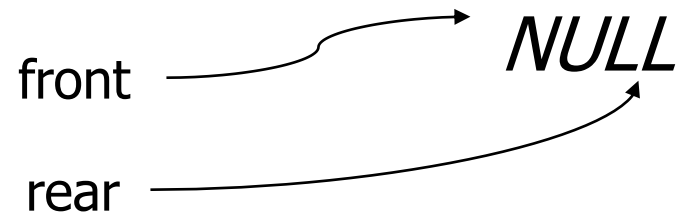
- Queue Class maintains two pointers
  - front: A pointer to the first element of the queue
  - rear: A pointer to the last element of the queue



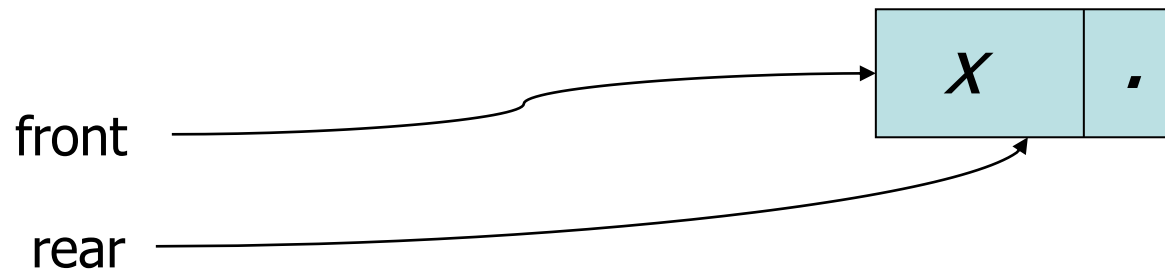
# Queue Operations (1)

---

- `MAKENULL(Q)`



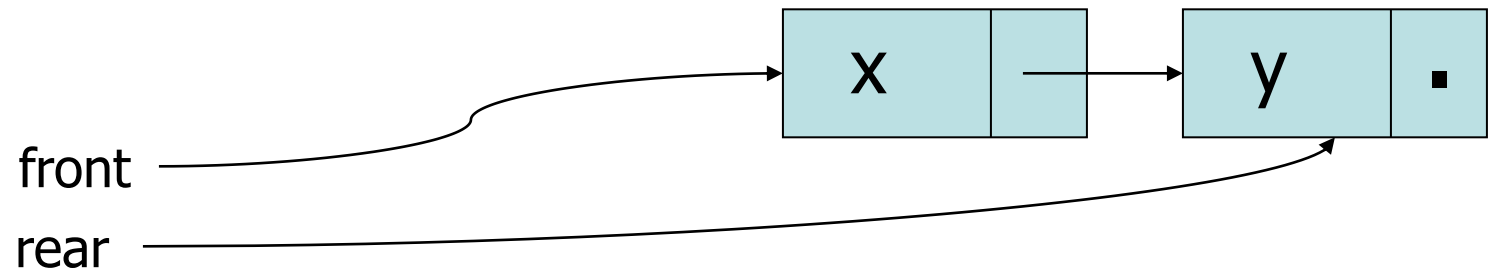
- `ENQUEUE (x, Q)`



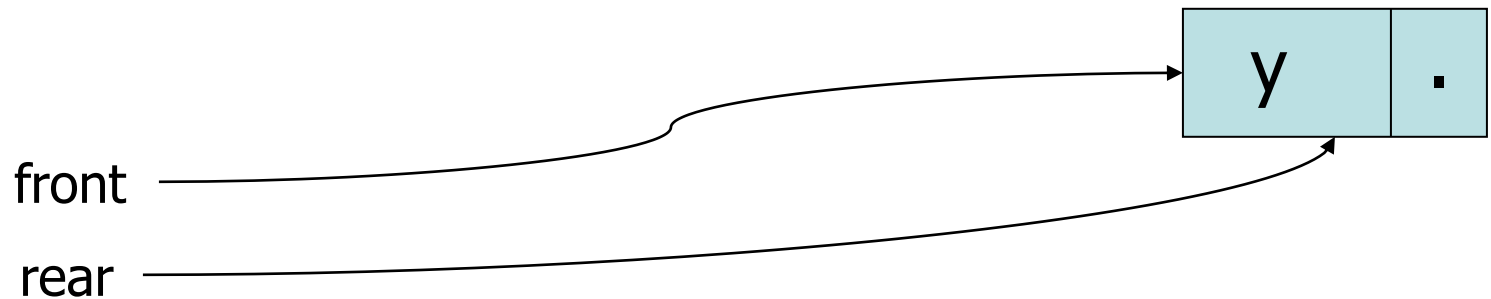
# Queue Operations

---

- ENQUEUE(y, Q)



- DEQUEUE (Q)



# Pointer Implementation – Code (1)

---

```
class DynIntQueue
{
    private:
        struct QueueNode // Structure to define linked list node
        {
            int value;
            QueueNode *next;
        };
        QueueNode *front; // pointer to the first node
        QueueNode *rear;  // pointer to the last node
        int numItems;      // Number of nodes in the linked list
    public:
        DynIntQueue(void);
        ~DynIntQueue(void);
        void enqueue(int);
        int dequeue(void);
        bool isEmpty(void);
        void makeNull(void);
};
```

# Pointer Implementation – Code (2)

---

- Constructor

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

- isEmpty() returns true if the queue is full and false otherwise

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

## Array Implementation – Code (3)

---

- Function enqueue inserts the value in num at the end of 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++;
}
```



## Array Implementation – Code (4)

---

- Function `dequeue` removes and returns the value at the front of the Queue

```
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--;
        if(!numItems) rear = NULL;
    }
    return num;
}
```

# Pointer Implementation – Code (5)

---

- Destructor

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

- `makeNull()` resets front & rear indices to NULL and sets `numItems` to 0

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

# Using Queues

```
void main(void)
{
    DynIntQueue iQueue;

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

    // Dequeue 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;
    }
}
```

## Output:

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

# Any Question So Far?

---

