National University of Computer & Emerging Sciences

Queues 2

Queues

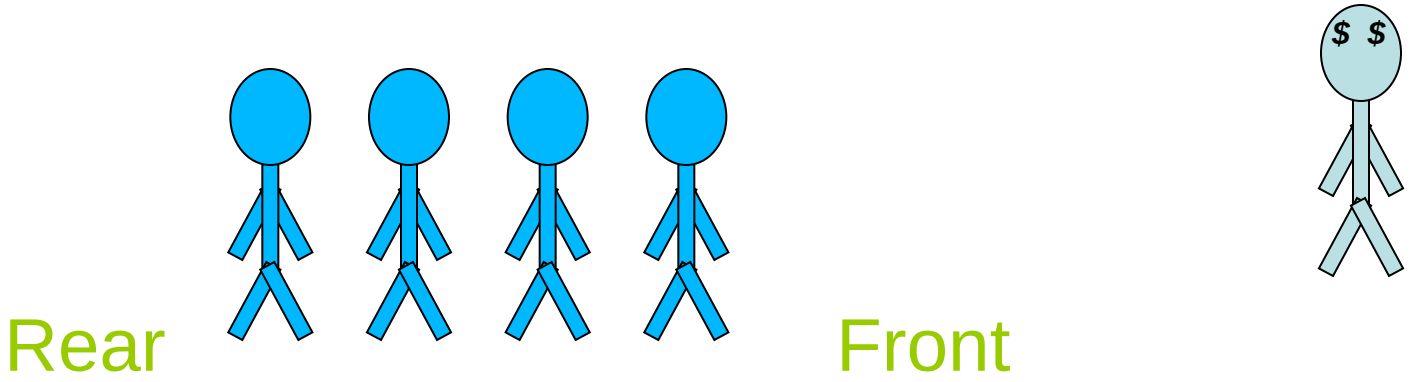
"A ***Queue*** is a special kind of list, where items are inserted at one end (***the rear***) And deleted at the other end (***the front***)"

Other Name:

- First In First Out (FIFO)

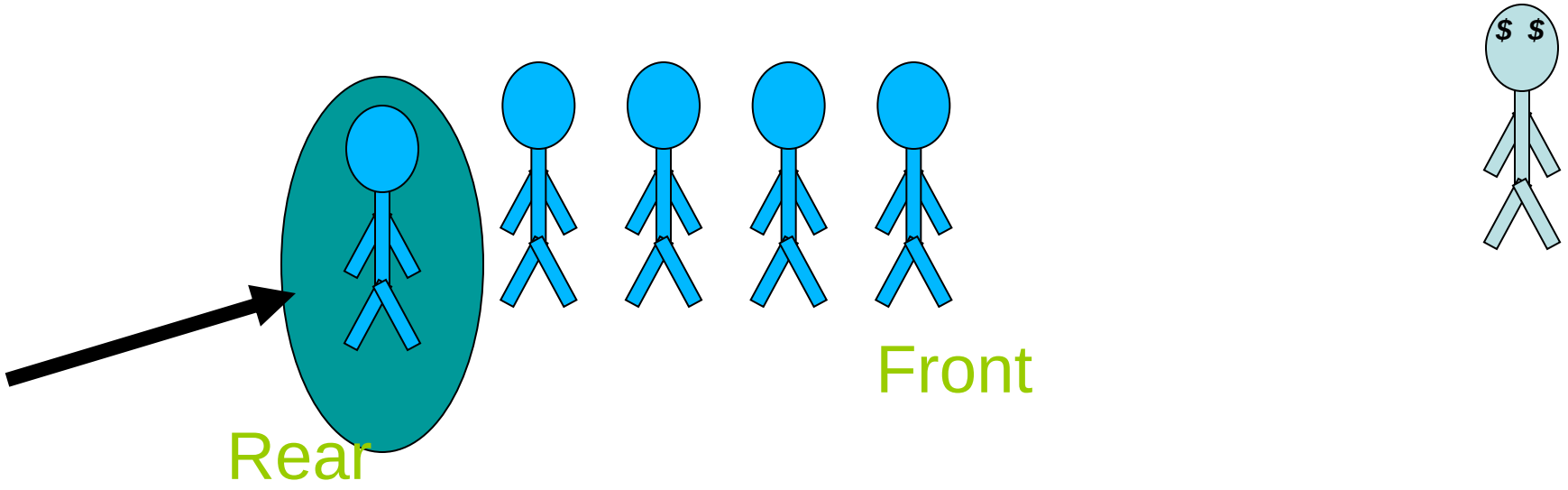
Queues

- A queue is like a line of people waiting for a bank teller. The queue has a **front** and a **rear**.



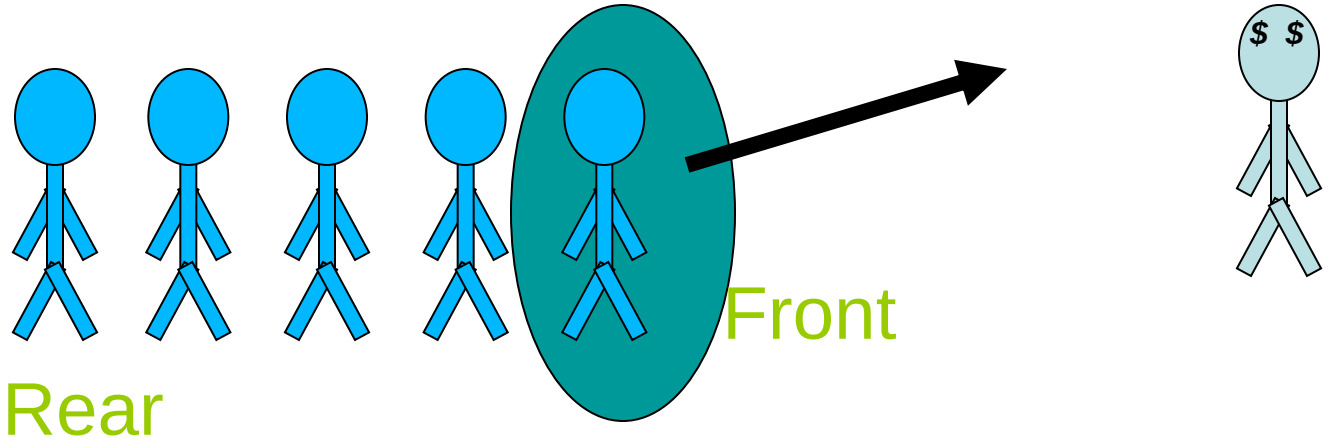
Queues

- New people must enter the queue at the rear.



Queues

- When an item is taken from the queue, it always comes from the front.



Common Operations (Queue ADT)

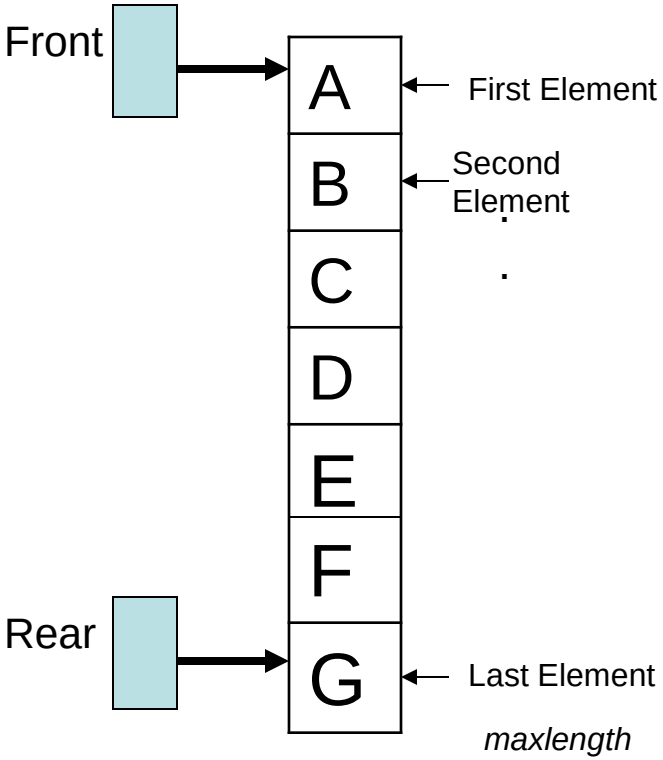
1. **MAKENULL(Q):** Makes Queue Q be an empty list.
2. **FRONT(*Q*):** Returns the first element on Queue Q.
3. **ENQUEUE(*x*,*Q*):** Inserts element x at the end of Queue Q.
4. **DEQUEUE(*Q*):** Deletes the first element of *Q.*
5. **EMPTY(*Q*):** Returns true if and only if Q is an empty queue.

Implementation

- Static
  - Queue is implemented by an array, and size of queue remains fix
- Dynamic
  - A **queue** can be **implemented** as a **linked list**, and *expand* or *shrink* with each *enqueue* or *dequeue* operation.

Alternative Array Implementation

- Use two counters that signify rear and front

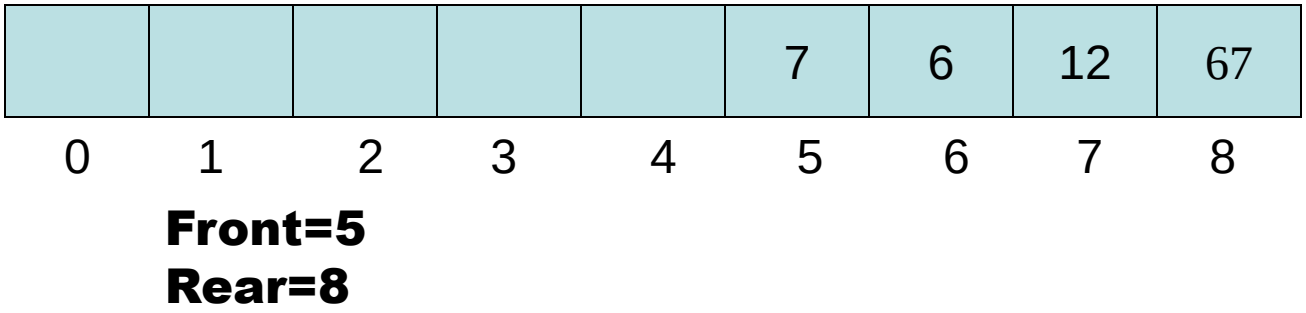


When queue is empty both front and rear are set to -1

While enqueueing increment rear by 1, and while dequeueing increment front by 1

When there is only one value in the Queue, both rear and front have same index

Array Implementation



How can we insert more elements? Rear index can not move beyond the last element….

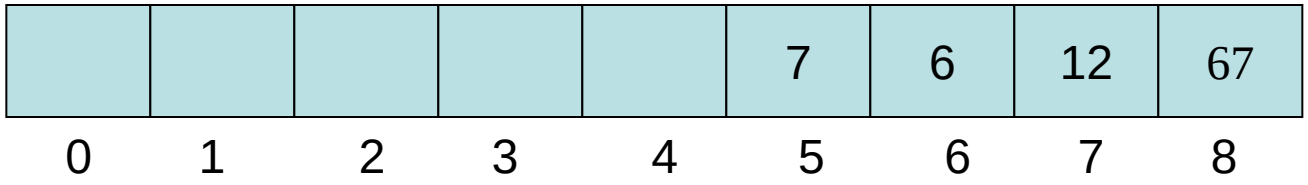
Solution: Using circular queue

- Allow rear to wrap around the array.

```
if(rear == queueSize-1)
        rear = 0;
else
        rear++;
```

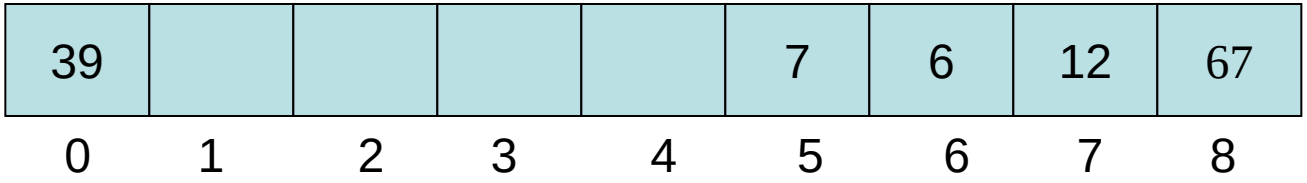
- Or use module arithmetic

```
rear = (rear + 1) % queueSize;
```



Front=5
Rear=8

Enqueue 39 **Rear=(Rear+1) mod Queue Size = (8+1) mod 9 = 0**



Front=5
Rear=0

How to determine empty and full Queues?

- It can be somewhat tricky
- Number of approaches
  - A counter indicating number of values in the queue can be used (We will use this approach)
  - We will see another approach as well at the end

Another implementation - using Arrays

```
class CQueue
{
    int *Data,QueueSize,Front,Rear;
public:
    CQueue(int size);
    ~CQueue(int size);
    bool IsFull();
    bool IsEmpty();
    void Enqueue(int num);
    int Dequeue();
    void MakeNull;
};
```

```
CQueue::CQueue(int size)
{
    Front=Rear=-1;
    Data=new int[size];
    QueueSize = size;
}
void CQueue ::Enqueue(int num);
{
    if ( IsFull() )
        {cout<<“Overflow”; return;}

    if (IsEmpty())
        Front=0;

    Rear=(Rear+1) % QueueSize;

    Data[Rear] = num;
}
```

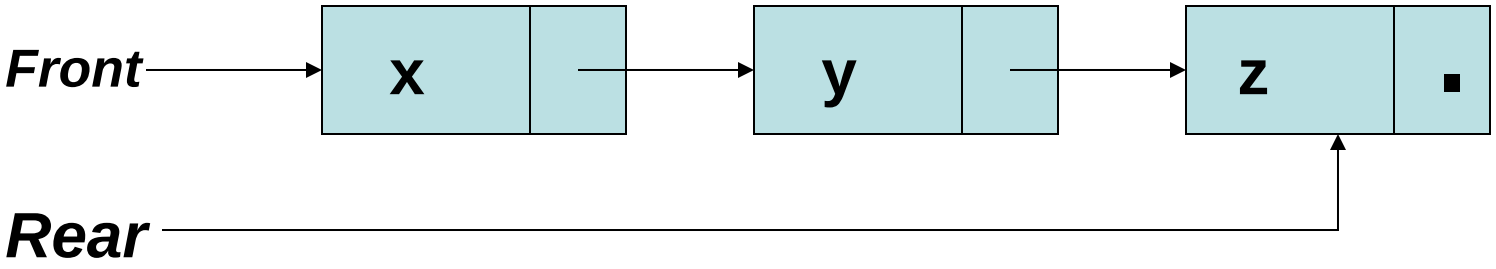
```
bool CQueue ::Dequeue(int &ReturnValue)
{
    if ( IsEmpty() )
        {cout<<“Underflow”; return false;}

    ReturnValue = Data[Front];

    if ( Front == Rear ) //only one element in the queue
        Front = Rear = -1;
    else
        Front = (Front+1) % QueueSize;

    return true;
}
```

```
bool CQueue::IsEmpty()
{
    if (Front==-1)
        return true; // we can check “Rear” too
    else
        return false;
}
bool CQueue::IsFull()
{
    If ( ( (Rear+1)%QueueSize ) == Front )
        return true;
    else
        return false;
}
```

A pointer Implementation of Queues

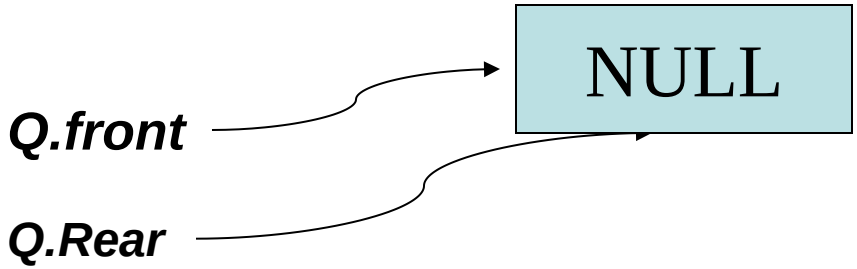
Keep two pointers:

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

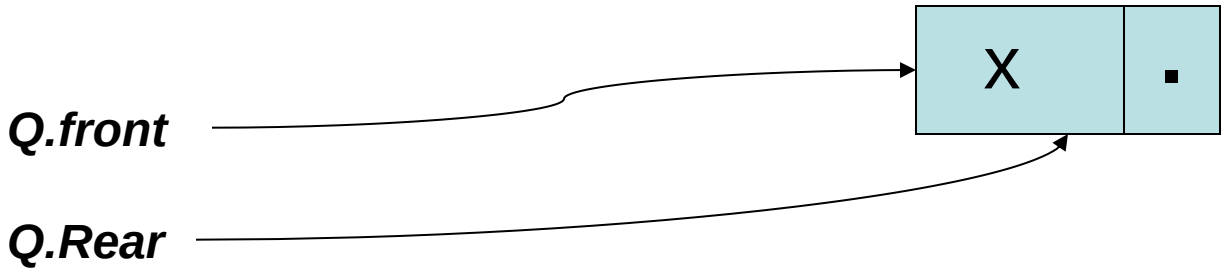


A pointer Implementation of Queues

MAKENULL(Q)

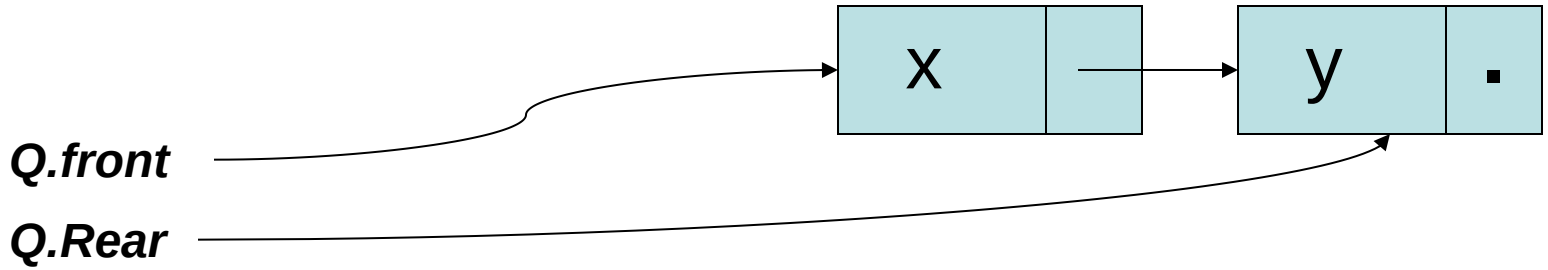


ENQUEUE(x,Q)

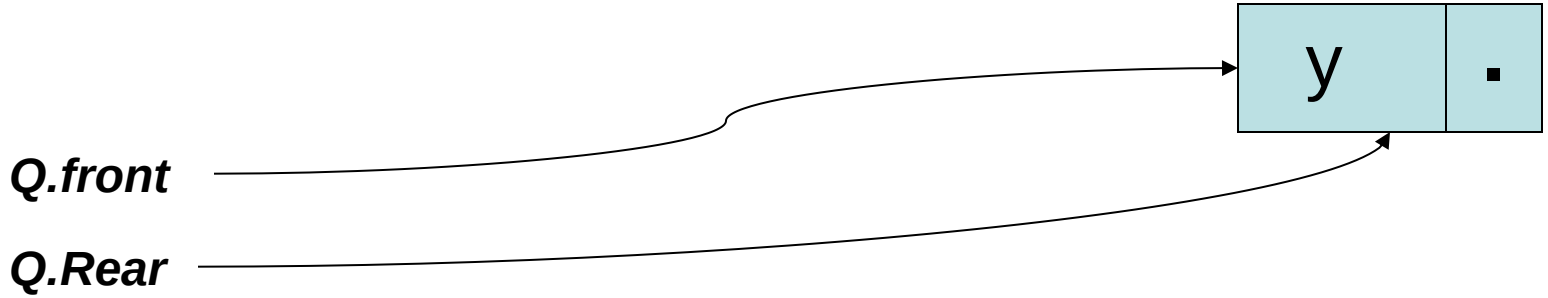


A pointer Implementation of Queues

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
```