DATA STRUCTURES AND ALGORITHMS

DR SAMABIA TEHSIN

BS (AI)

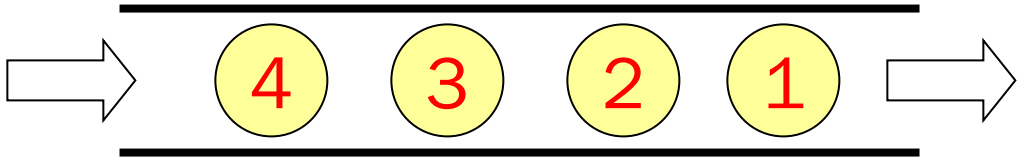
What is a queue?

It is an ordered group of homogeneous items of elements.

Queues have two ends:

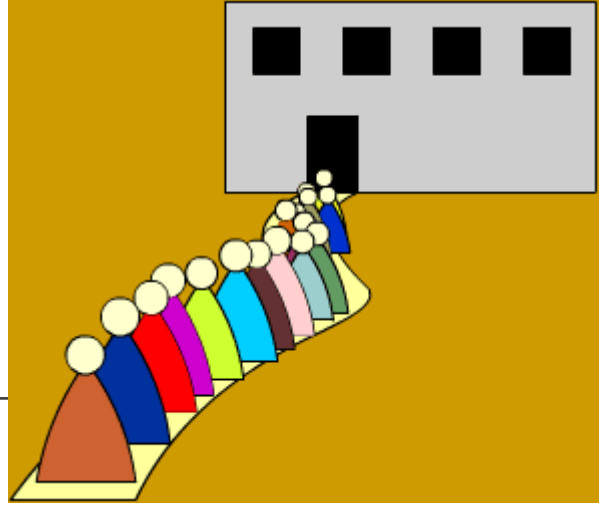
- Elements are added at one end.(Rear)
- Elements are removed from the other end.(Front)

The element added first is also removed first (**FIFO**: First In, First Out).



no changes of order

Queue Specification



Definitions: (provided by the user)

- *MAX_ITEMS*: Max number of items that might be on the queue
- *ItemType*: Data type of the items on the queue

- Operations
  - MakeEmpty
  - Boolean IsEmpty
  - Boolean IsFull
  - Enqueue (ItemType newItem)
  - Dequeue (ItemType& item) (serve and retrieve)

Enqueue (ItemType newItem)

Function: Adds newItem to the rear of the queue.

Preconditions: Queue has been initialized and is not full.

Postconditions: newItem is at rear of queue.

Dequeue (ItemType& item)

***Function*:** Removes front item from queue and returns it in item.

Preconditions: Queue has been initialized and is not empty.

Postconditions: Front element has been removed from queue and item is a copy of removed element.

Implementation issues

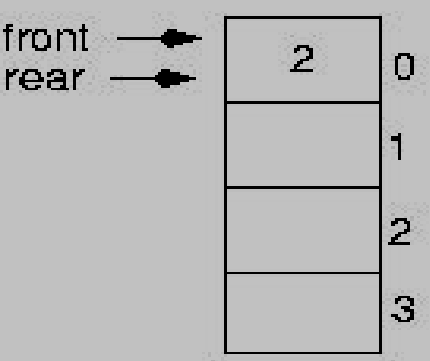
Implement the queue as a *circular structure*.

How do we know if a queue is full or empty?

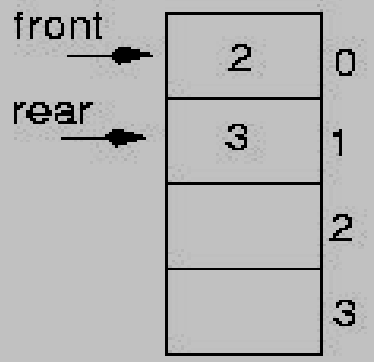
Initialization of *front* and *rear*.

Testing for a *full* or *empty* queue.

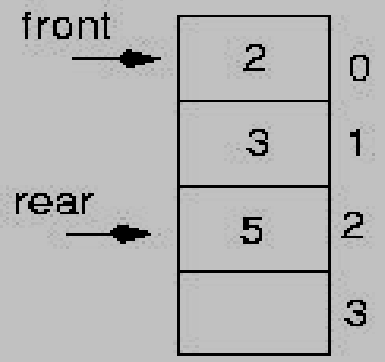
q.Enqueue(2)



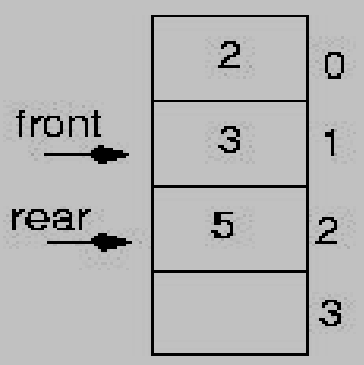
q.Enqueue(3)



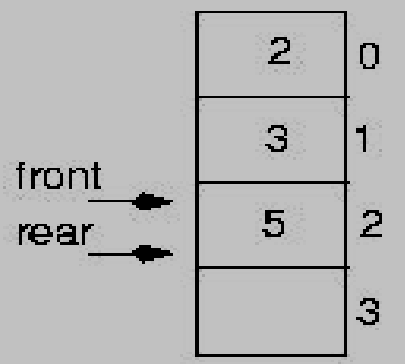
q.Enqueue(5)



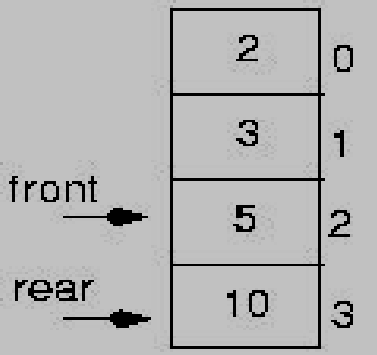
q.Dequeue(item)
item = 2



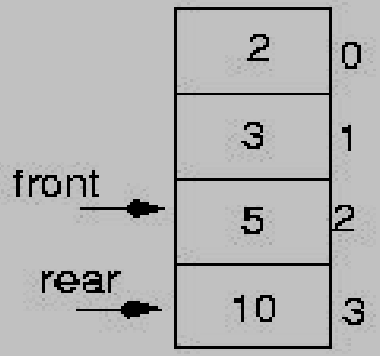
q.Dequeue(item)
item = 3



q.Enqueue(10)

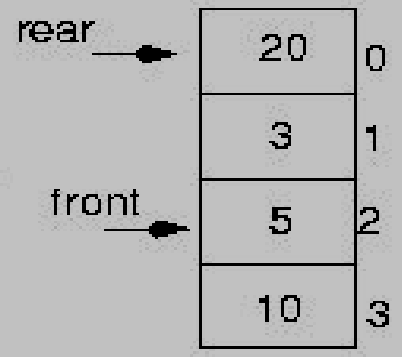


q.Enqueue(20) ???



Let the queue elements

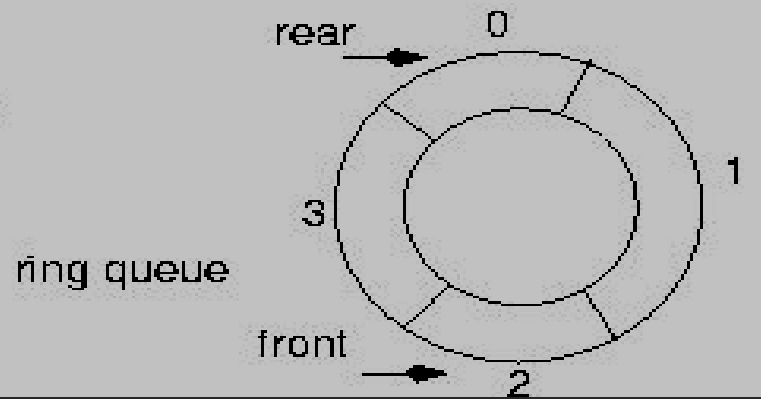
"wrap around"



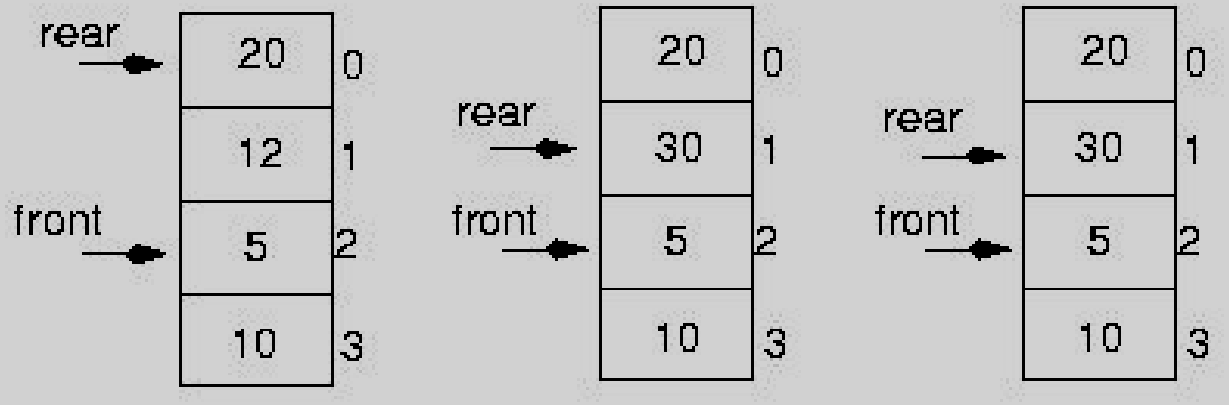

```
if(rear == maxQue -1)
  rear = 0;
else
  rear = rear + 1;
```

or

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



q.Enqueue(30) q.Enqueue(50) ???

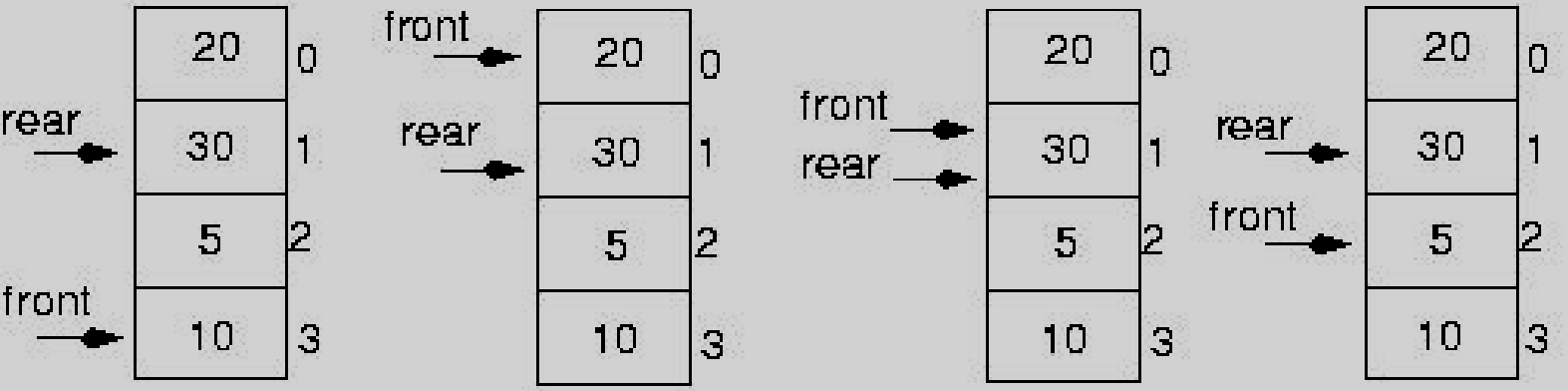


The queue is full !!

What is the condition for a full queue ?

```
rear + 1 == front
```

q.Dequeue(item) item = 5 q.Dequeue(item) item = 10 q.Dequeue(item) item = 20 q.Dequeue(item) item = 30



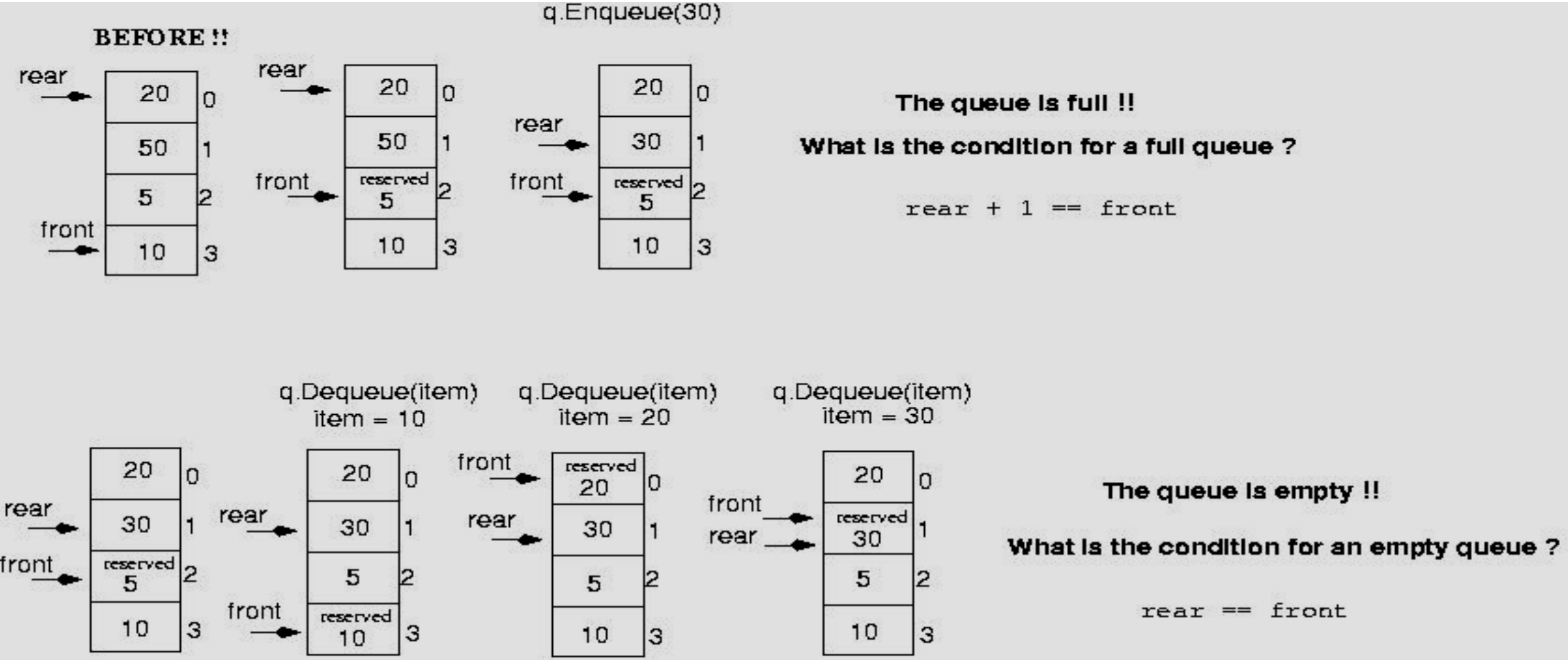
The queue is empty !!

What is the condition for an empty queue ?

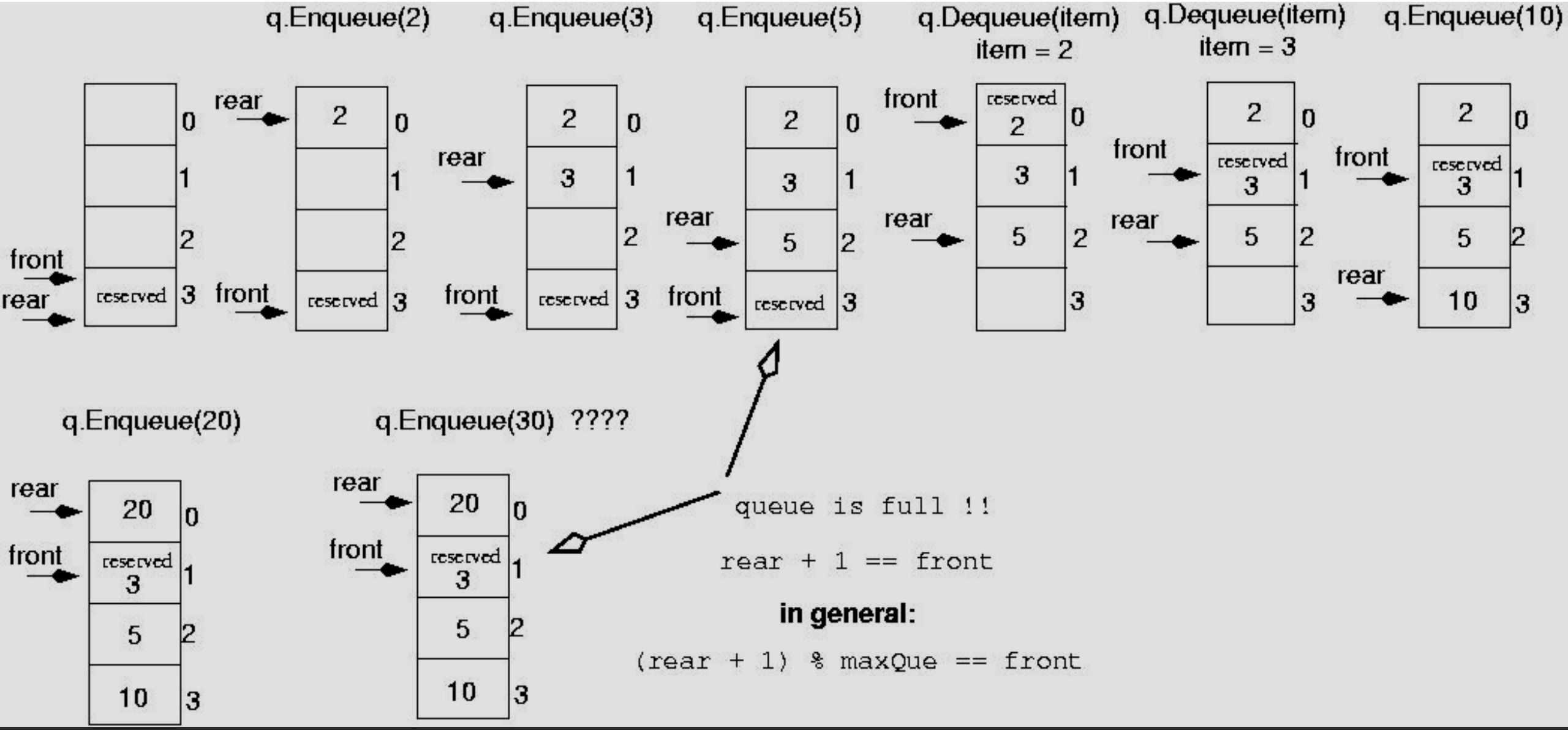
```
rear + 1 == front
```

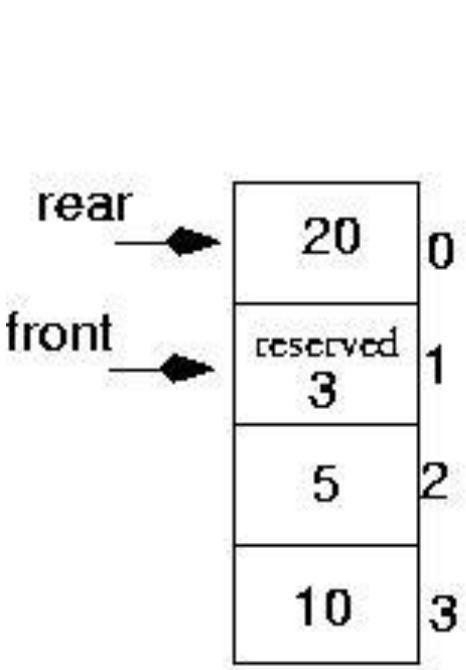
We cannot distinguish between the two cases !!!

Make *front* point to the element **preceding** the front element in the queue (one memory location will be wasted).

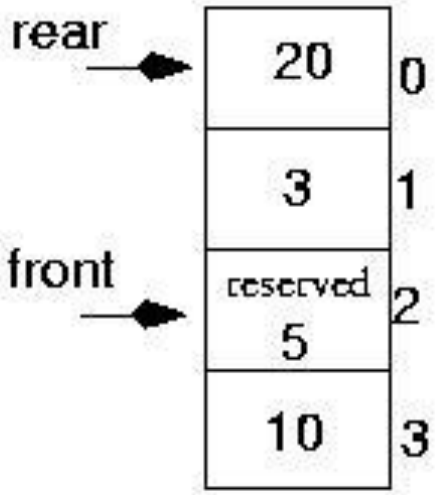


Initialize *front* and *rear*

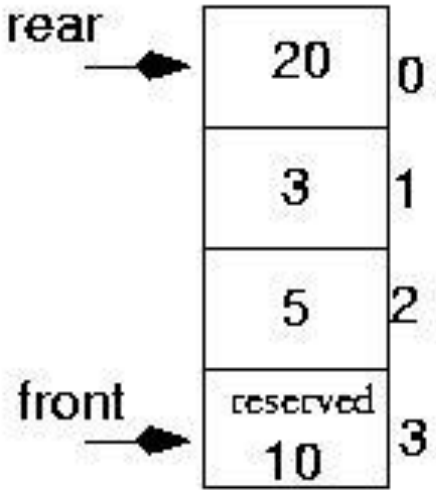




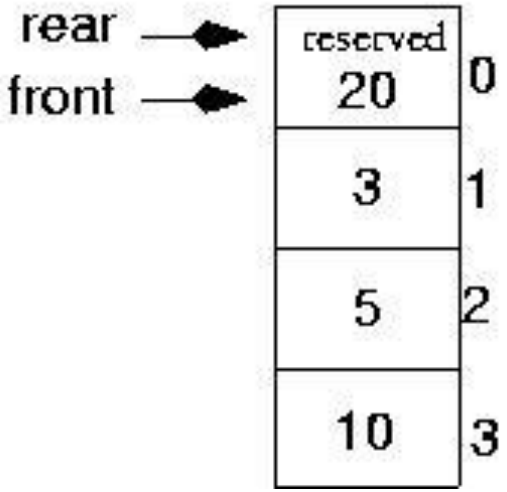
q.Dequeue(item)
item = 5



q.Dequeue(item)
item = 10



q.Dequeue(item)
item = 20



Queue is empty now!!

rear == front

Queue Implementation

```
template<class ItemType>
class QueueType {
 public:
    QueueType(int);
    QueueType();
    ~QueueType();
    void MakeEmpty();
    bool IsEmpty() const;
    bool IsFull() const;
    void Enqueue(ItemType);
    void Dequeue(ItemType&);
```

```
private:
    int front;
    int rear;
    ItemType* items;
    int maxQue;
};
```

Queue Implementation (cont.)

```
template<class ItemType>
QueueType<ItemType>::QueueType(int max)
{
 maxQue = max + 1;
 front = maxQue - 1;
 rear = maxQue - 1;
 items = new ItemType[maxQue];
}
```

Queue Implementation (cont.)

```
template<class ItemType>
QueueType<ItemType>::~QueueType()
{
 delete [] items;
}
```

Queue Implementation (cont.)

```
template<class ItemType>
void QueueType<ItemType>:: MakeEmpty()
{
 front = maxQue - 1;
 rear = maxQue - 1;
}
```

Queue Implementation (cont.)

```
template<class ItemType>
bool QueueType<ItemType>::IsEmpty() const
{
 return (rear == front);
}

template<class ItemType>
bool QueueType<ItemType>::IsFull() const
{
 return ( (rear + 1) % maxQue == front);
}
```

Queue Implementation (cont.)

```
template<class ItemType>
void QueueType<ItemType>::Enqueue (ItemType newItem)
{
 rear = (rear + 1) % maxQue;
 items[rear] = newItem;
}
```

Queue Implementation (cont.)

```
template<class ItemType>

void QueueType<ItemType>::Dequeue (ItemType& item)

{

 front = (front + 1) % maxQue;

 item = items[front];

}
```

Queue overflow

The condition resulting from trying to add an element onto a full queue.

```
if(!q.IsFull())
        q.Enqueue(item);
```

Queue underflow

The condition resulting from trying to remove an element from an empty queue.

```
if(!q.IsEmpty())
  q.Dequeue(item);
```

Example: recognizing palindromes

A *palindrome* is a string that reads the same forward and backward.

Able was I ere I saw Elba

We will read the line of text into both a stack and a queue.

Compare the contents of the stack and the queue character-by-character to see if they would produce the same string of characters.

Example: recognizing palindromes

Stack

Queue

Example: recognizing palindromes

```
#include <iostream.h>
#include <ctype.h>
#include "stack.h"
#include "queue.h“
int main()
{
 StackType<char> s;
 QueType<char> q;
 char ch;
 char sItem, qItem;
 int mismatches = 0;
```

Example: recognizing palindromes

```
while( (!q.IsEmpty()) && (!s.IsEmpty()) ) {
```

```
  s.Pop(sItem);
  q.Dequeue(qItem);

  if(sItem != qItem)
    ++mismatches;
}
if (mismatches == 0)
  cout << "That is a palindrome" << endl;
else
 cout << That is not a palindrome" << endl;

return 0;
}
```

Credits and Acknowledgements

Lectures by Prof. Yung Yi, KAIST, South Korea.

Lectures by **Selim Aksoy**, Bilkent University, Ankara, Turkey

Lecture slides by Dept of Computer Science, Boston University