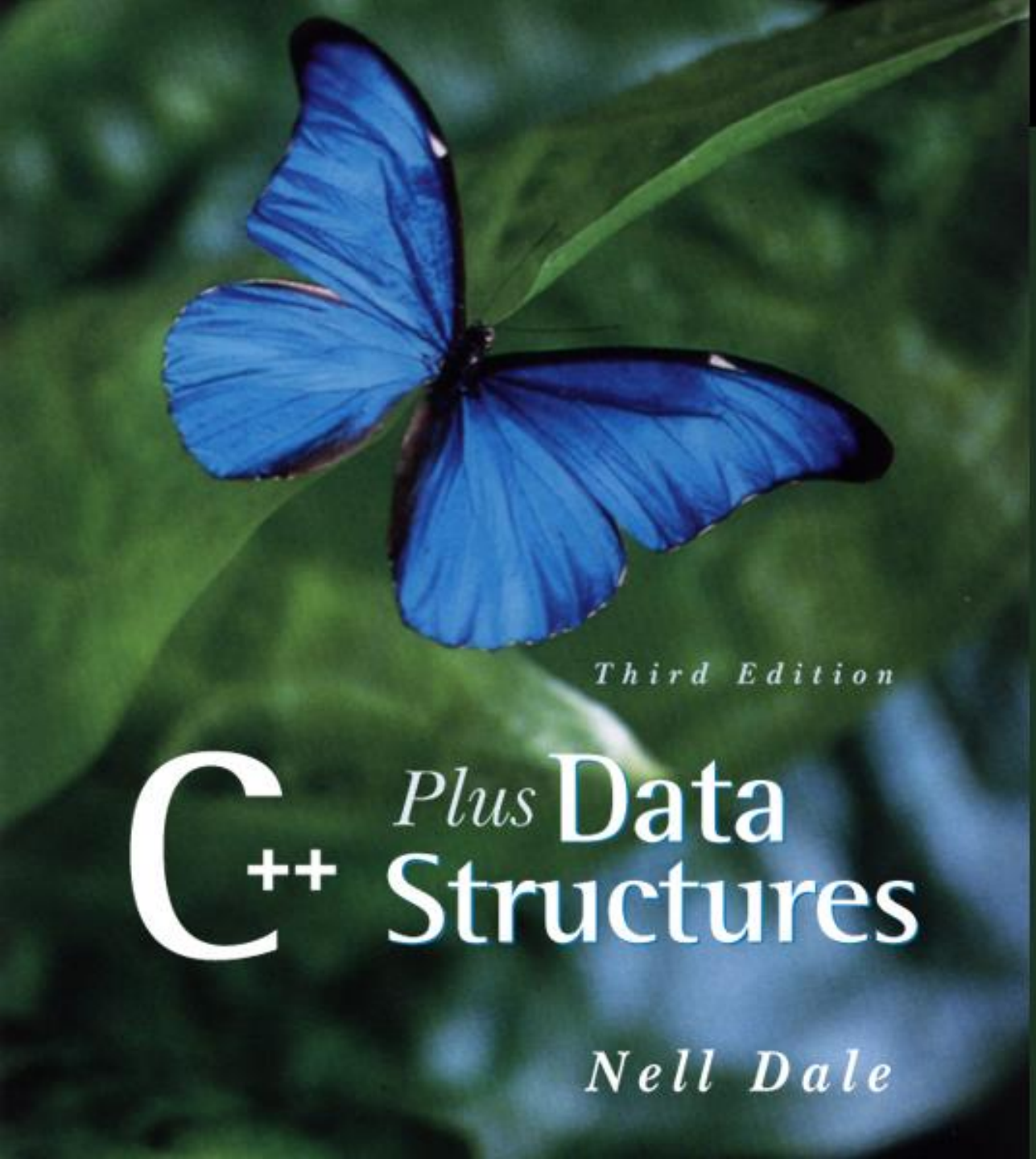


Chapter

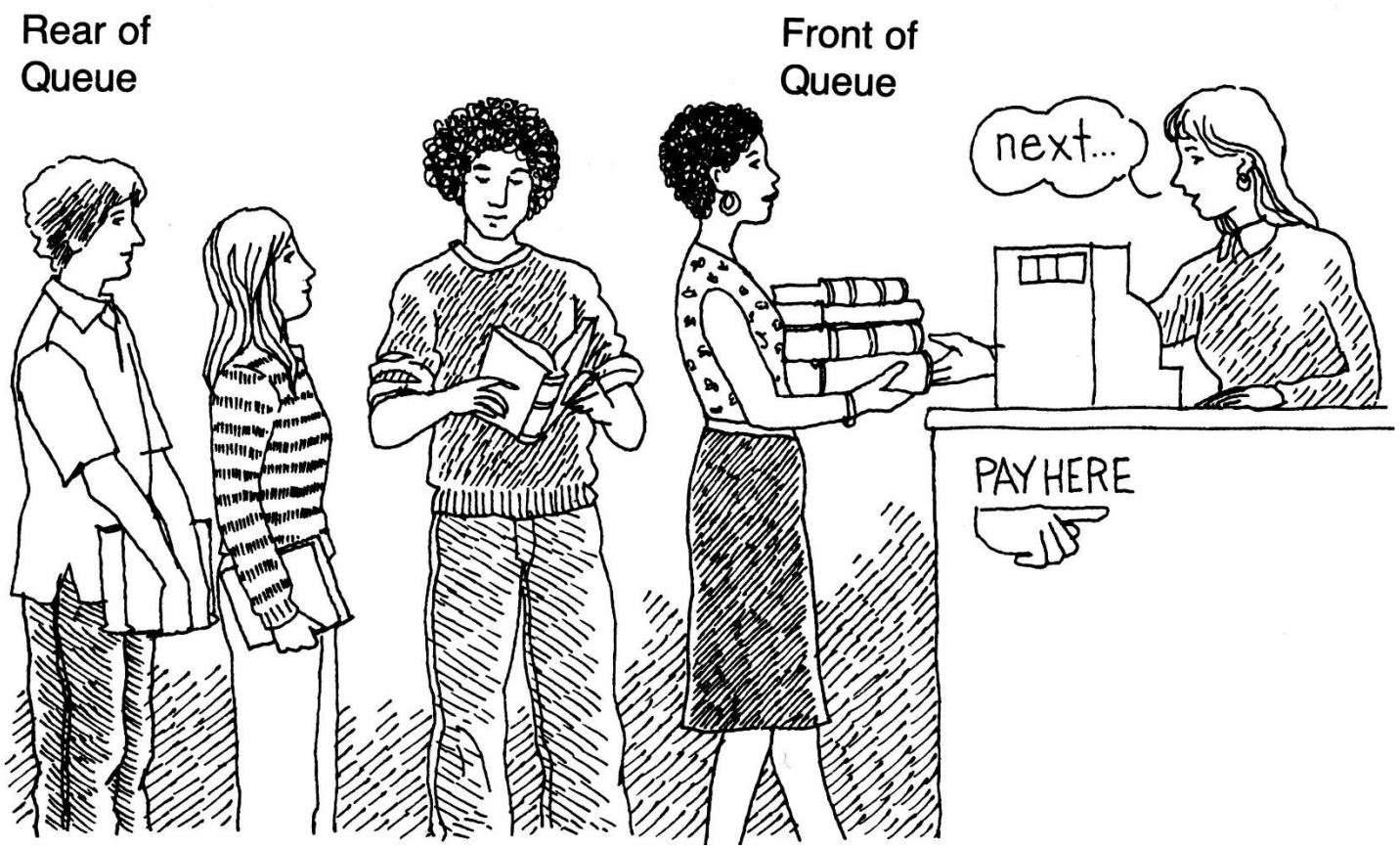
4

ADTs Stack and Queue

What is a Queue?

- ***Logical (or ADT) level:*** A queue is an ordered group of homogeneous items (elements), in which new elements are added at one end (the rear), and elements are removed from the other end (the front).
- A queue is a FIFO "first in, first out" structure.

Example: Queue of Customers



Enqueue (ItemType newItem)

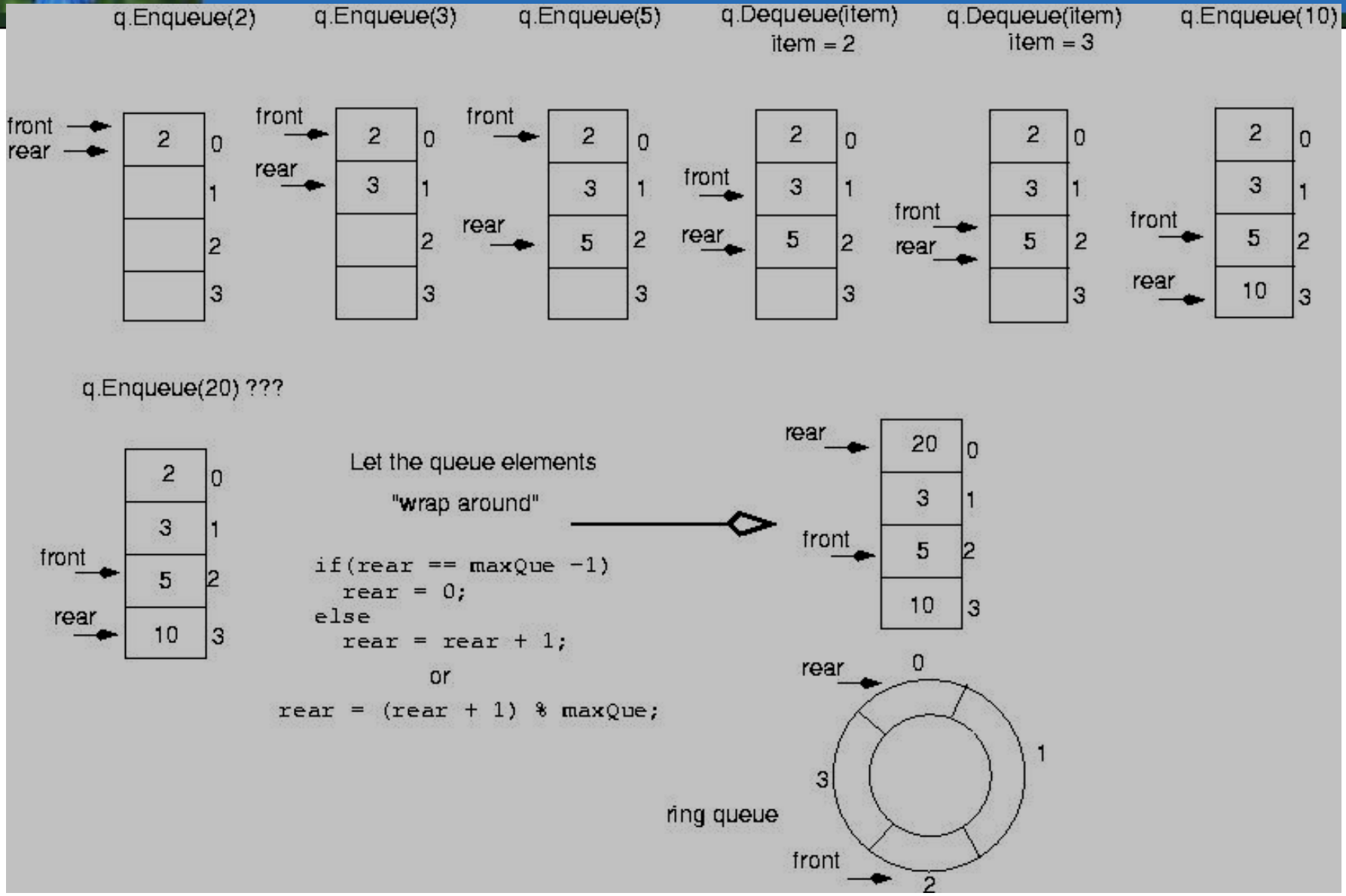
- *Function*: Adds newItem to the rear of the queue.
- *Preconditions*: Queue has been initialized and is not full.
- *Postconditions*: newItem is at rear of queue.

Dequeue (ItemType& item)

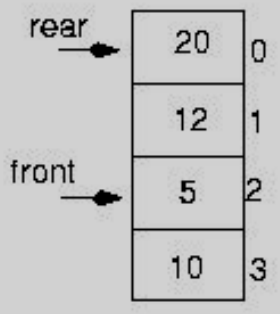
- *Function*: Removes front item from queue and returns it in item.
- *Preconditions*: Queue has been initialized and is not empty.
- *Postconditions*: Front element has been removed from queue and item is a copy of removed element.

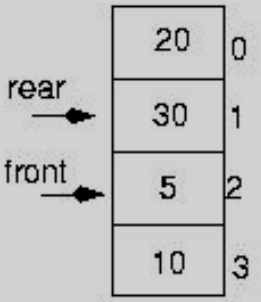
Implementation issues

- 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.

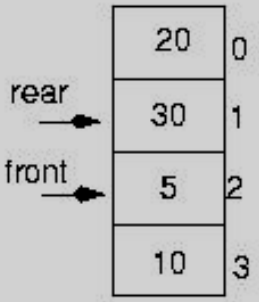

q.Enqueue(2)
q.Enqueue(3)
q.Enqueue(5)
q.Dequeue(item)
item = 2
q.Dequeue(item)
item = 3
q.Enqueue(10)
front
rear
2
3
5
10
0
1
2
3
q.Enqueue(20) ???
Let the queue elements
"wrap around"
if(rear == maxQue -1)
rear = 0;
else
rear = rear + 1;
or
rear = (rear + 1) % maxQue;
20
ring queue

q.Enqueue(30)





q.Enqueue(50) ???

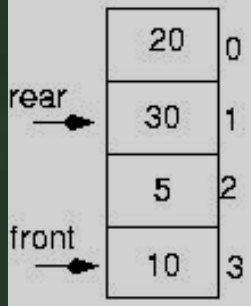


The queue is full !!

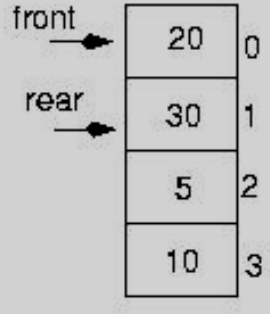
What is the condition for a full queue ?

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

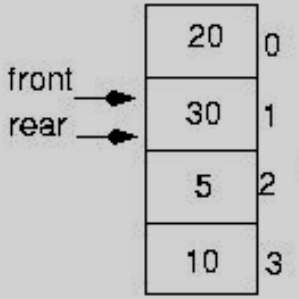
q.Dequeue(item)
item = 5



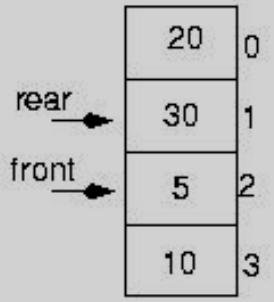
q.Dequeue(item)
item = 10



q.Dequeue(item)
item = 20



q.Dequeue(item)
item = 30

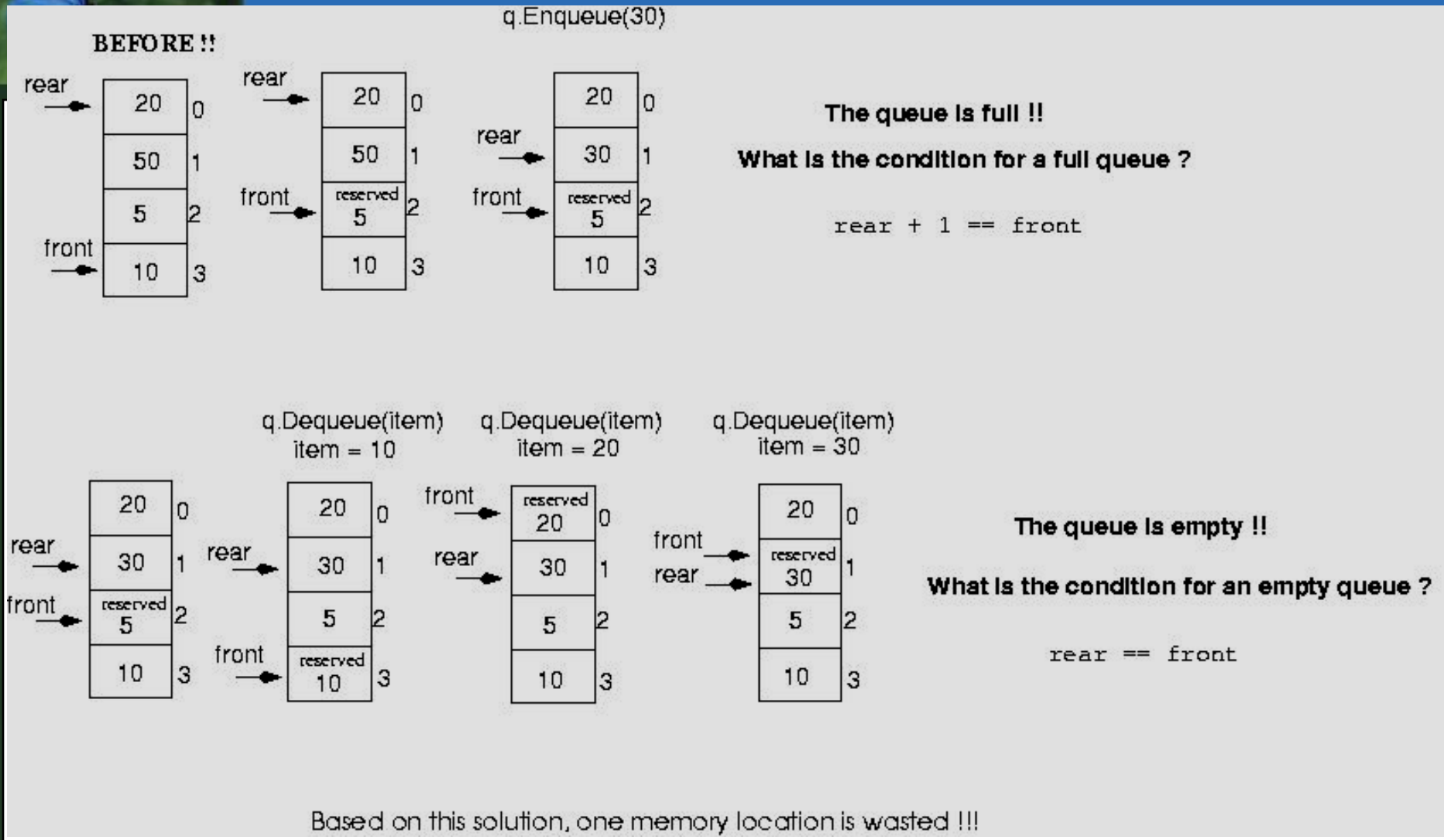


The queue is empty !!

What is the condition for an empty queue ?

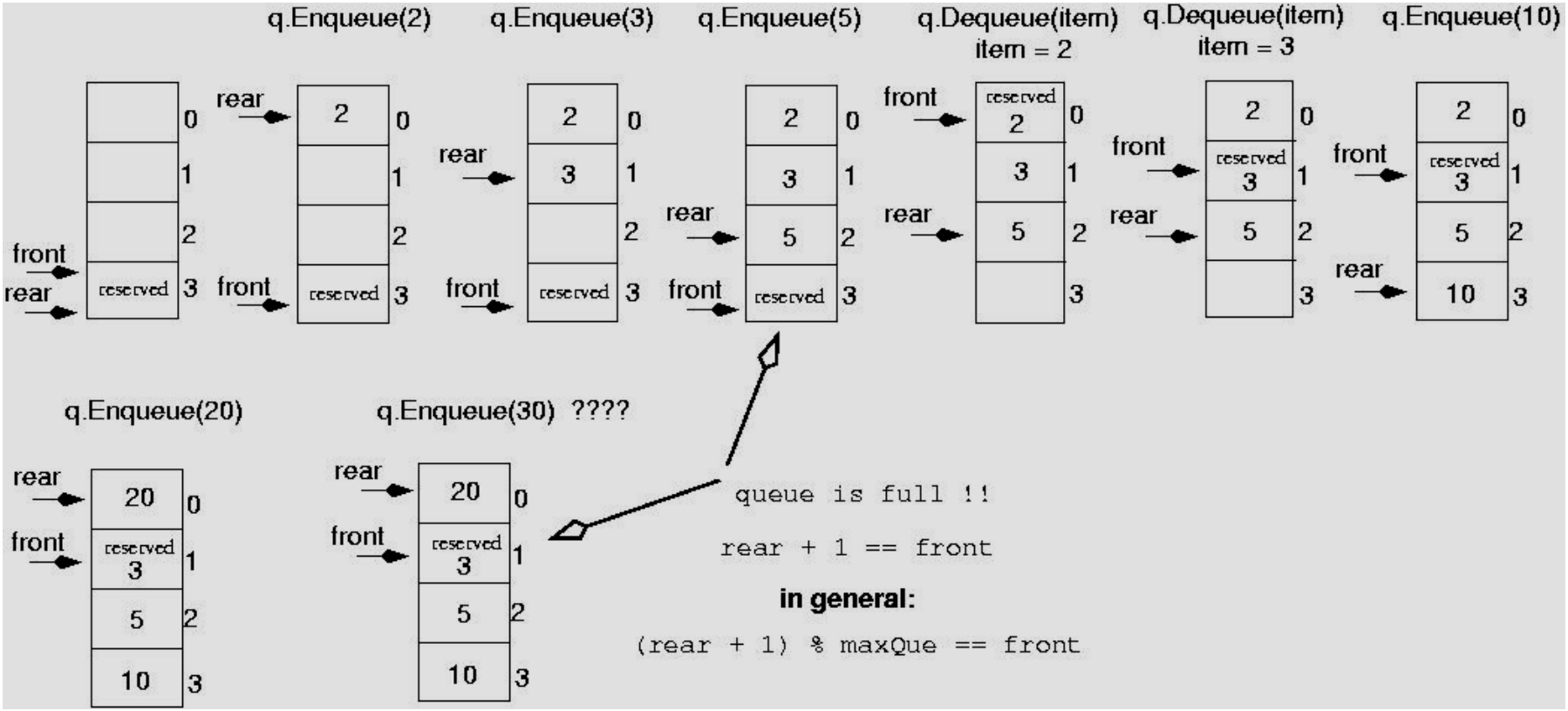
```
rear + 1 == front
```

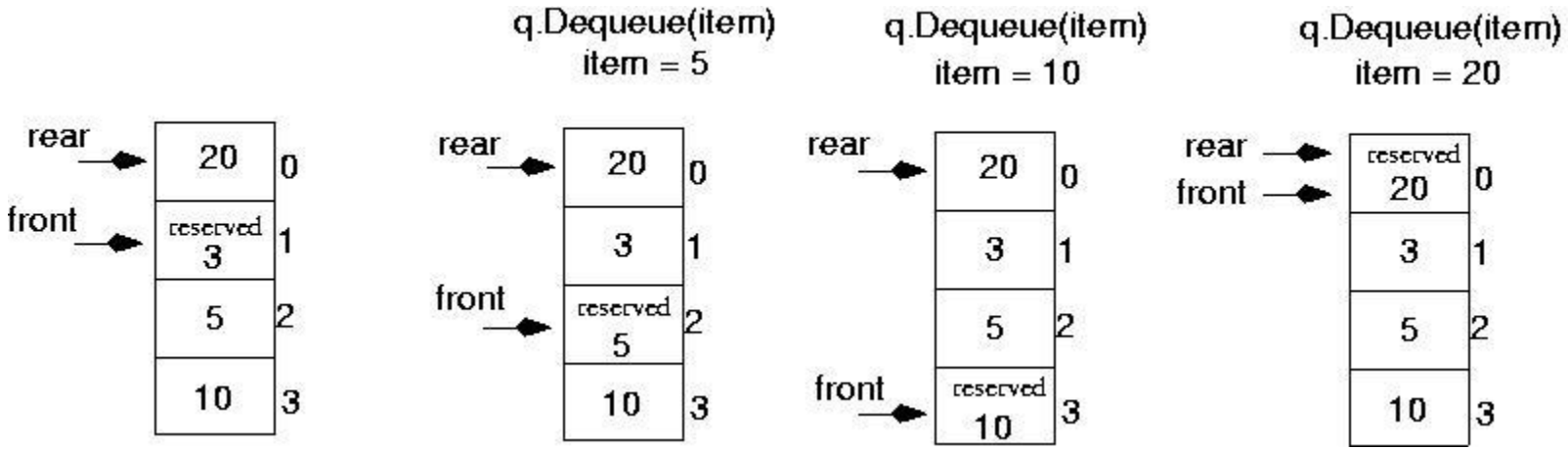
We cannot distinguish between the two cases !!!



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

Initialize *front* and *rear*





Queue is empty now!!

rear == front

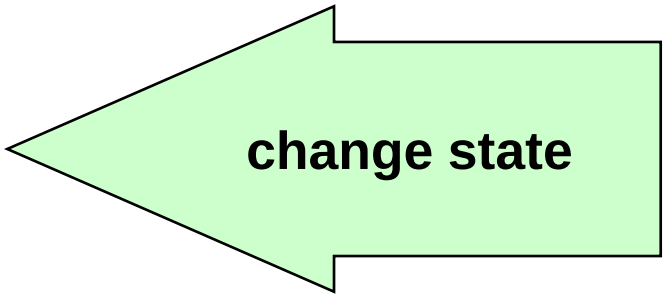
Queue ADT Operations

- **MakeEmpty -- Sets queue to an empty state.**
- **IsEmpty -- Determines whether the queue is currently empty.**
- **IsFull -- Determines whether the queue is currently full.**
- **Enqueue (ItemType newItem) -- Adds newItem to the rear of the queue.**
- **Dequeue (ItemType& item) -- Removes the item at the front of the queue and returns it in item.**

ADT Queue Operations

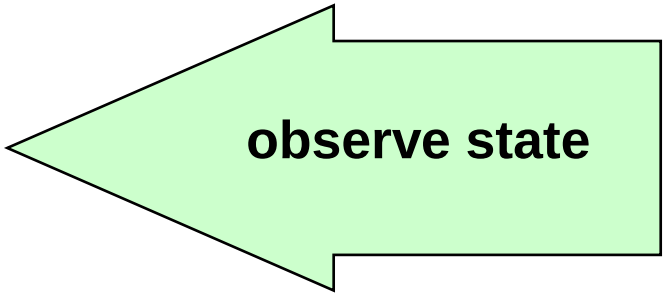
Transformers

- MakeEmpty
- Enqueue
- Dequeue

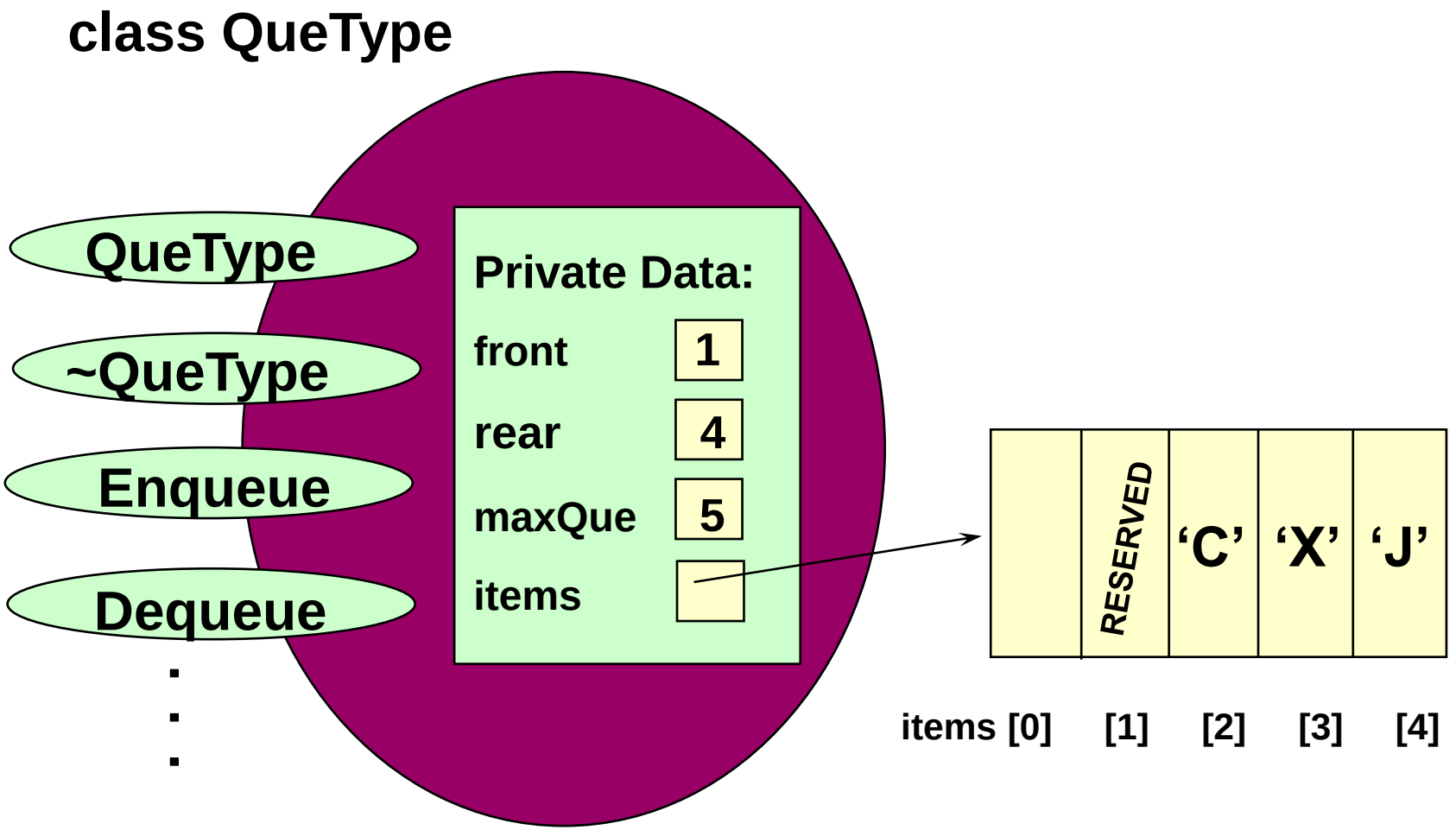


Observers

- IsEmpty
- IsFull



DYNAMIC ARRAY IMPLEMENTATION



```
//-----------------------------------------------------------
// CLASS TEMPLATE DEFINITION FOR CIRCULAR QUEUE
#include "ItemType.h"          // for ItemType
template<class ItemType>
class QueType
{
public:
   QueType( );
   QueType( int max );         // PARAMETERIZED CONSTRUCTOR
   ~QueType( ) ;               // DESTRUCTOR
   . . .
   bool IsFull( ) const;
   void Enqueue( ItemType item );
   void Dequeue( ItemType&  item );
private:
   int        front;
   int        rear;
   int        maxQue;
   ItemType*  items;     // DYNAMIC ARRAY IMPLEMENTATION };
```

```
//-----------------------------------------------------
// CLASS TEMPLATE DEFINITION FOR CIRCULAR QUEUE  cont'd
//-----------------------------------------------------

template<class ItemType>
QueType<ItemType>::QueType( int max )    // PARAMETERIZED
{
   maxQue = max + 1;
   front = maxQue - 1;
   rear = maxQue - 1;
   items = new ItemType[maxQue];    // dynamically allocates
}

template<class ItemType>
bool QueType<ItemType>::IsEmpty( )

{
   return ( rear == front )
}
```

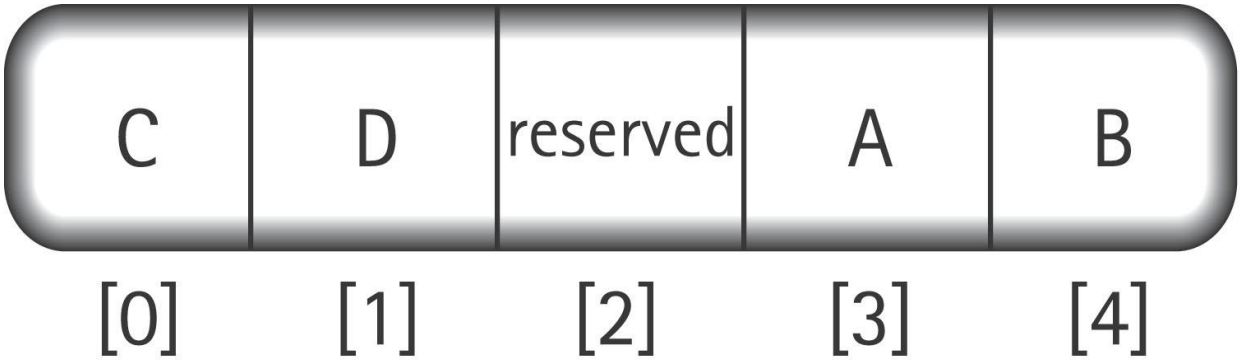
```
//---------------------------------------------------------
// CLASS TEMPLATE DEFINITION FOR CIRCULAR QUEUE  cont'd
//---------------------------------------------------------

template<class ItemType>
QueType<ItemType>::~QueType( )
{
   delete [ ] items;          // deallocates array
}

.
.
.

template<class ItemType>
bool QueType<ItemType>::IsFull( )

{                                                  // WRAP AROUND
   return ( (rear + 1) % maxQue == front )
}
```

```
//-----------------------------------------------------
// CLASS TEMPLATE DEFINITION FOR CIRCULAR QUEUE  cont'd
//-----------------------------------------------------

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

template<class ItemType>
void QueType<ItemType>::Dequeue(ItemType &item)
{
   front = (front+1) % maxQue;
   item = items[front];
}
```

SAYS ALL PUBLIC MEMBERS OF QueType CAN BE INVOKED FOR OBJECTS OF TYPE CountedQuType

```
// DERIVED CLASS CountedQueType FROM BASE CLASS QueType

template<class ItemType>
class CountedQueType : public QueType<ItemType>
{
public:
   CountedQueType( );
   void Enqueue( ItemType newItem );
   void Dequeue( ItemType&  item );
   int LengthIs( ) const;
   // Returns number of items on the counted queue.

private:
   int length;
};
```

class CountedQueType<char>




```
front = 2
rear = 1
```

```
// Member function definitions for class CountedQue

template<class ItemType>
CountedQueType<ItemType>::CountedQueType( ) : QueType<ItemType>( )
{
   length = 0 ;
}


template<class ItemType>
int CountedQueType<ItemType>::LengthIs( ) const
{
   return  length ;
}
```

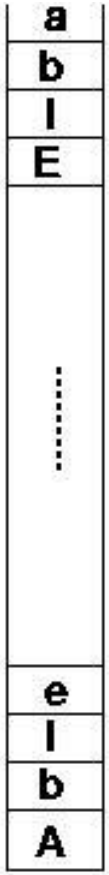
```
template<class ItemType>
void CountedQueType<ItemType>::Enqueue( ItemType newItem )
       // Adds newItem to the rear of the queue.
       // Increments length.
{
   length++;

   QueType<ItemType>::Enqueue( newItem );
}

template<class ItemType>
void CountedQueType<ItemType>::Dequeue(ItemType& item )
       // Removes item from the rear of the queue.
       // Decrements length.
{
   length--;

   QueType<ItemType>::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.



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;
```

```
cout << "Enter string: " << endl;

while(cin.peek() != '\\n') {

  cin >> ch;
  if(isalpha(ch)) {

    if(!s.IsFull())
      s.Push(toupper(ch));

    if(!q.IsFull())
      q.Enqueue(toupper(ch));
  }
}
```

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;
}
```