THE CLASS ARRAYQUEUE

THE CLASS ARRAYQUEUE

- **Efficiently using an array**

backIndex = (backIndex + 1);

frontIndex = (frontIndex + 1);

THE CLASS ARRAYQUEUE

- **Efficiently using an array**
  - Implement it as a ***Circular Array***

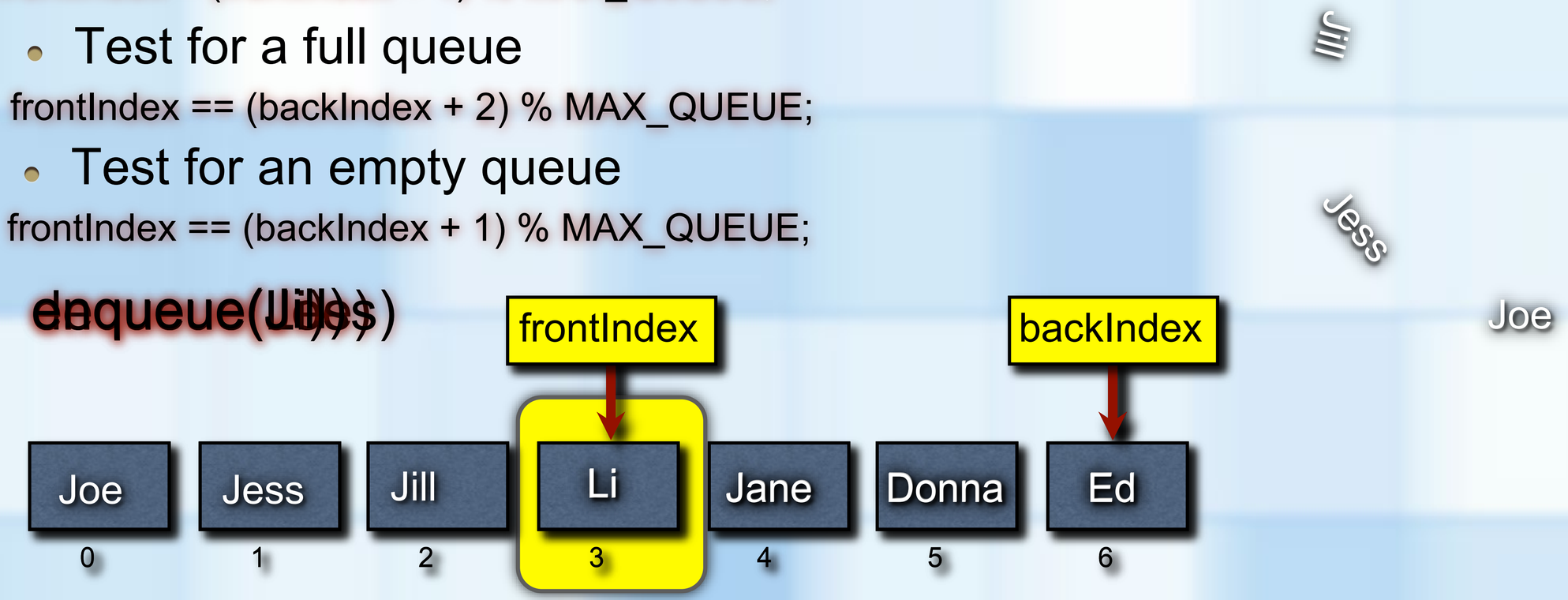
```
backIndex = (backIndex + 1) % MAX_QUEUE;
frontIndex = (frontIndex + 1) % MAX_QUEUE;
```

  - Test for a full queue

```
frontIndex == (backIndex + 2) % MAX_QUEUE;
```

  - Test for an empty queue

```
frontIndex == (backIndex + 1) % MAX_QUEUE;
```



The Class ArrayQueue

- **ArrayClass Implementation**
  - Data Fields
  - Constructor

ArrayQueue.h

```
template<class ItemType>
class ArrayQueue : public QueueInterface<ItemType>
{
private:
  static const int DEFAULT_CAPACITY = 5;
  ItemType items[DEFAULT_CAPACITY];
  int frontIndex;
  int backIndex;
  int count;

public:
   ArrayQueue();

   bool isEmpty() const noexcept;
   bool enqueue(const ItemType& someItem);
   bool dequeue();

   ItemType peekFront() const;
}; // end ArrayQueue
```

ArrayQueue.cpp

```
template<class ItemType>
ArrayQueue<ItemType>::ArrayQueue()
    : front(0),
      back(DEFAULT_CAPACITY - 1),
      count(0)
{  }  // end default constructor
```

Pearson

THE CLASS ARRAYQUEUE

- **ArrayClass Implementatio**
  - Data Fields
  - Constructor
  - peekFront
  - enqueue
  - dequeue

```
template<class ItemType>
ItemType ArrayQueue<ItemType>::peekFront() const
{
   if (isEmpty())
     throw PrecondViolatedExcept("peekFront() called with empty queue");

   // Queue is not empty; return front
   return items[frontIndex];
}  // end peekFront
```

```
template<class ItemType>
bool ArrayQueue<ItemType>::enqueue(const ItemType& someItem) noexcept
{
   bool result = false;
   if (count < MAX_QUEUE)
   {
     // Queue has room for another item
     backIndex = (backIndex + 1) % MAX_QUEUE;
     items[backIndex] = someItem;
     count++;
      result = true;
  }  // end if

  return result;
}  // end enqueue
```

Pearson

THE CLASS ARRAYQUEUE

- **ArrayClass Implementation**
  - Data Fields
  - Constructor
  - peekFront
  - enqueue
  - dequeue

```
template<class ItemType>
bool ArrayQueue<ItemType>::dequeue()
{
    bool result = false;
    if (!isEmpty())
    {
        result = true;
      frontIndex = (frontIndex + 1) % MAX_QUEUE;
      count--;
        result = true;
   }  // end if

    return result;
}  // end dequeue
```

The Class LinkedQueue

THE CLASS LINKEDQUEUE

- **Implementing the ADT Queue**
  - Using a linked chain
- **How can we add and remove nodes efficiently?**
  - dequeue at the head of the chain
  - enqueue at the tail of the chain

```
template<class ItemType>
class LinkedQueue : public QueueInterface<ItemType>
{
private:
   Node<ItemType>* frontPtr;
   Node<ItemType>* backPtr;

public:
   LinkedQueue();
   LinkedQueue  (const LinkedQueue& aQueue);
   ~LinkedQueue();

    bool isEmpty() const noexcept;
    bool enqueue(const ItemType& someItem);
    bool dequeue();
    ItemType peekFront() const;
}; // end LinkedQueue
```

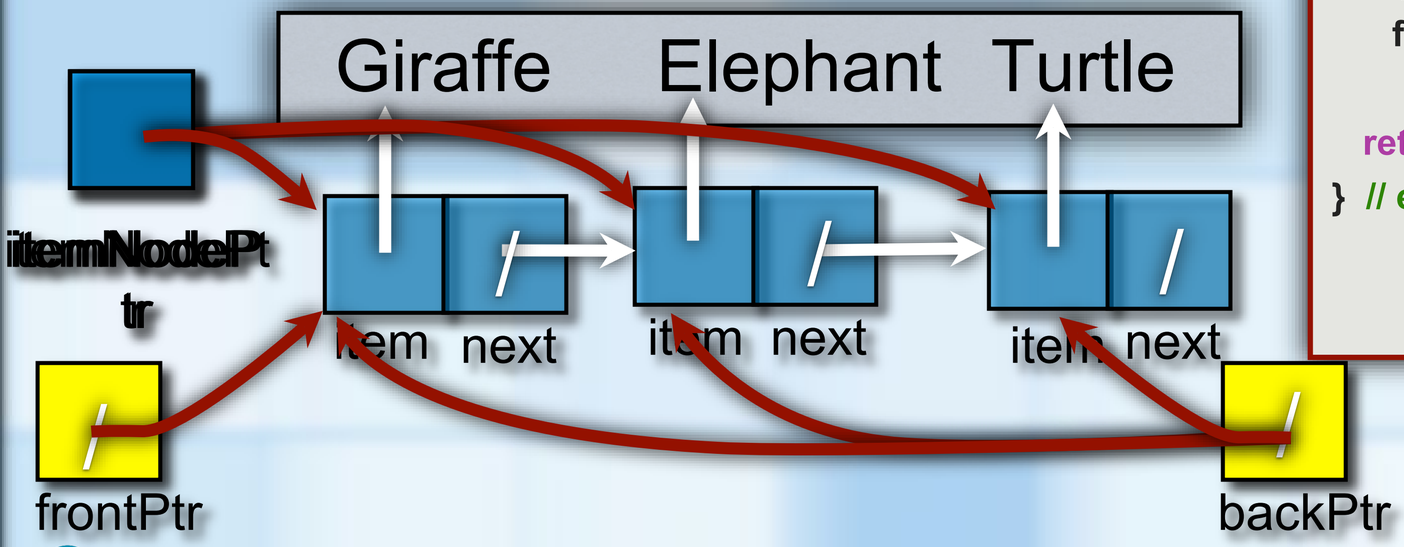
frontPtr

backPtr

Pearson

THE CLASS LINKEDQUEUE -- enqueue

- **Implementing the ADT Queue**
  - Using a linked chain
- **How can we add and remove nodes efficiently?**
  - dequeue at the head of the chain
  - enqueue at the tail of the chain

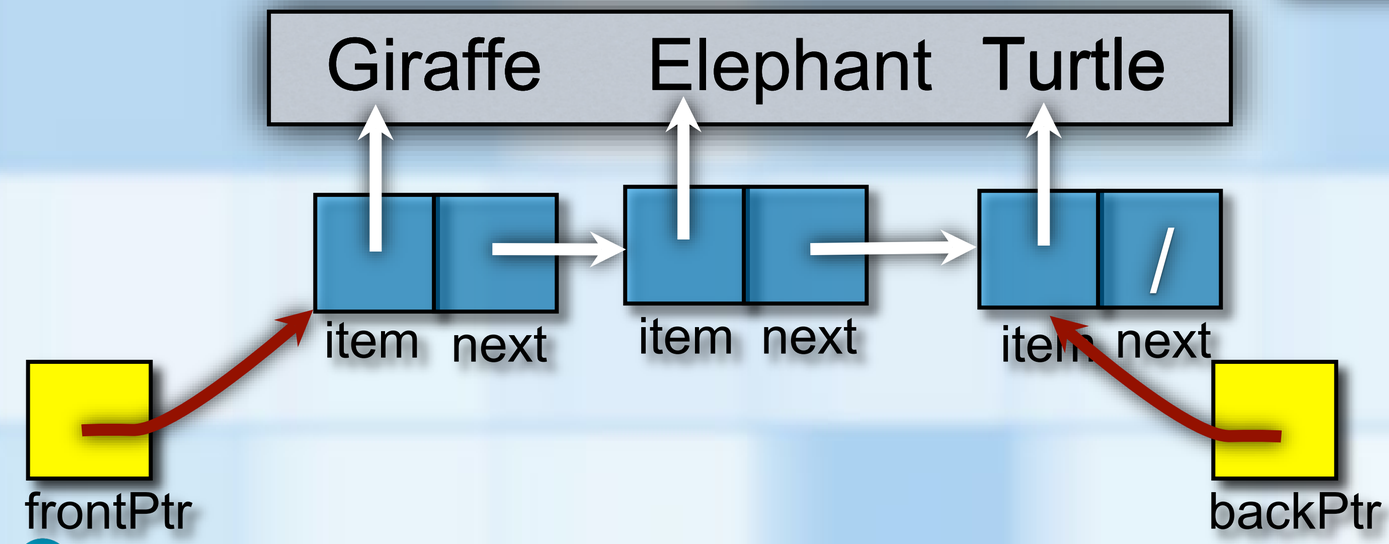


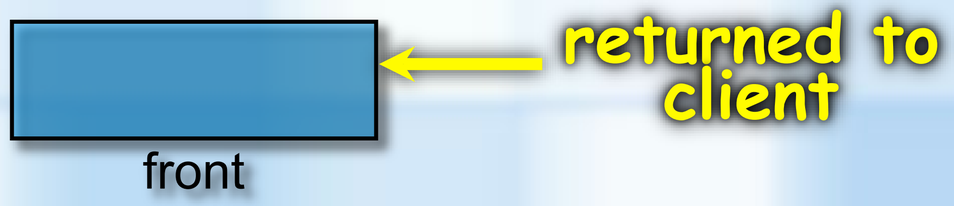

```
template<class ItemType>
bool LinkedQueue<ItemType>::
                    enqueue(const ItemType& someItem)
{
  auto itemNodePtr =
          std::make_shared<Node<ItemType>>(someItem);

  backPtr->setNext(itemNodePtr);
  backPtr = itemNodePtr;
  if (isEmpty())
    frontPtr = itemNodePtr;

  return true;
}  // end enqueue
```

THE CLASS LINKEDQUEUE -- peekFront

- **Implementing the ADT Queue**
  - Using a linked chain
- **How can we add and remove nodes efficiently?**
  - dequeue at the head of the chain
  - enqueue at the tail of the chain

```
template<class ItemType>
ItemType LinkedQueue<ItemType>::peekFront() const
{
  if (isEmpty())
    throw PrecondViolatedExcep(
                "peekFront() called with empty queue.");

  return frontPtr->getItem();
}  // end peekFront
```

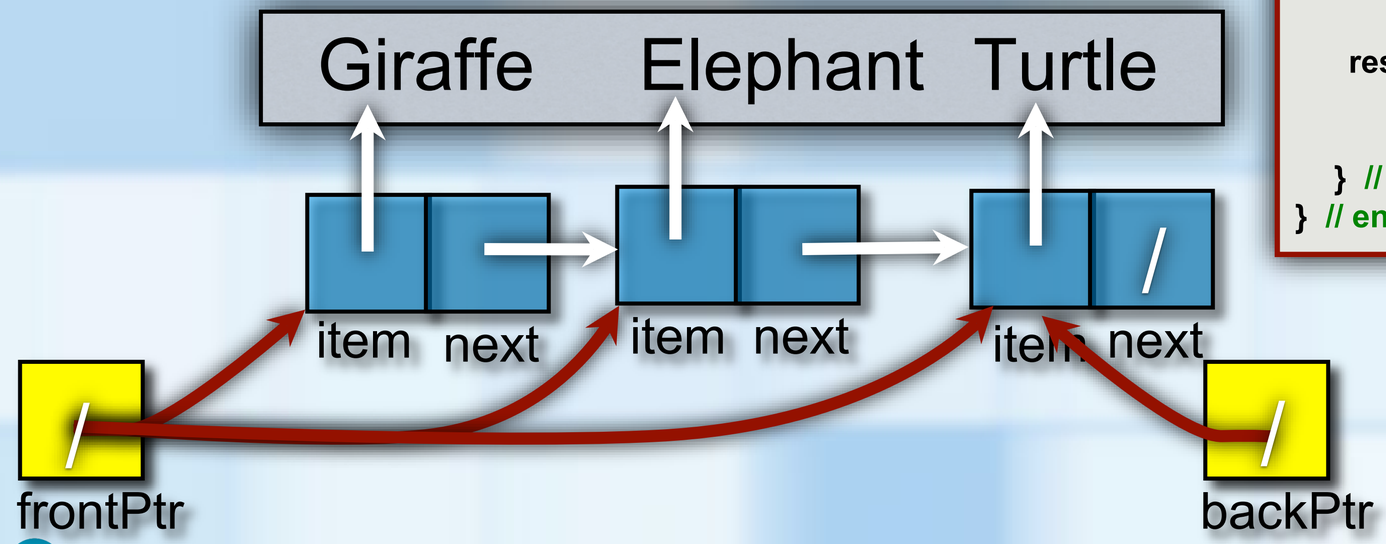




Pearson

THE CLASS LINKEDQUEUE -- dequeue

- **Implementing the ADT Queue**
  - Using a linked chain
- **How can we add and remove nodes efficiently?**
  - dequeue at the head of the chain
  - enqueue at the tail of the chain




```
template<class ItemType>
bool LinkedQueue<ItemType>::dequeue()
{
    bool result = false;
    if (!isEmpty())
    {

     frontPtr = frontPtr->getNext();

     if (frontPtr == backPtr)
     {
       backPtr = nullptr;
     }


     result = true;


    }  // end if
}  // end dequeue
```

The Class LinkedStack -- isEmpty

- **Implementing the ADT Stack**
  - Using a linked chain
- **Which end is the top of the stack?**
  - Using the head is faster than other nodes

```
template<class ItemType>
bool LinkedQueue<ItemType>::isEmpty() const
{
   return (backPtr == nullptr) || (frontPtr == nullptr);
}  // end isEmpty
```

frontPtr

backPtr

Pearson