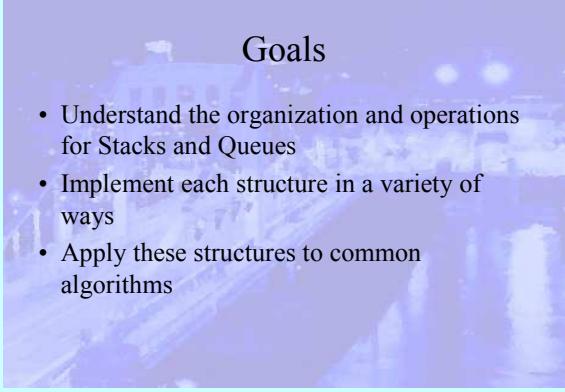


**Data Structures
and Algorithms I**

Stacks and Queues

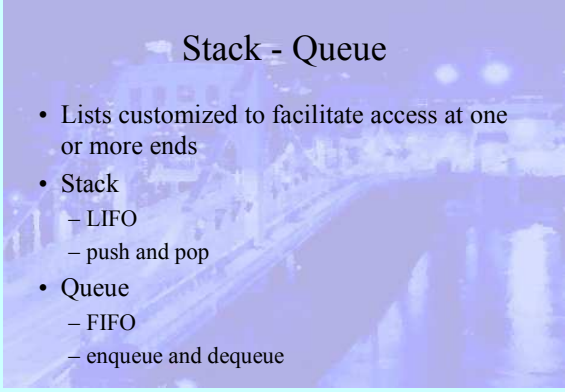
Dr. Tim Margush
University of Akron
© 2008



Goals

- Understand the organization and operations for Stacks and Queues
- Implement each structure in a variety of ways
- Apply these structures to common algorithms

© 2008 Dr. Tim Margush



Stack - Queue

- Lists customized to facilitate access at one or more ends
- Stack
 - LIFO
 - push and pop
- Queue
 - FIFO
 - enqueue and dequeue

© 2008 Dr. Tim Margush

Stack – Array Implementation

```
public class ArrayStack<E>{  
    private Object [] elts = new Object[5];  
    private int top=0;  
    public E push(E item){}  
    public E pop(){}  
    public E top(){}  
    boolean isFull(){}  
    boolean isEmpty(){}  
    ...  
}
```

© 2008 Dr. Tim Margush

Push – Array Implementation

```
public class ArrayStack<E>{  
    private Object [] elts = new Object[5];  
    private int top=0;  
    public E push(E item){  
  
    }  
}
```

© 2008 Dr. Tim Margush

Pop – Array Implementation

```
public class ArrayStack<E>{  
    private Object [] elts = new Object[5];  
    private int top=0;  
    public E pop(){  
  
    }  
}
```

© 2008 Dr. Tim Margush

Uses

- Recursion
- Parsing
- Infix to Postfix
- RPN evaluation
- Depth First Search

© 2008 Dr. Tim Margush

RPN Evaluation

3 7 5 - 9 + *

© 2008 Dr. Tim Margush

RPN Evaluation

```
Iterator<Token> e = expression.iterator();
Stack<Token> pending = new Stack<Token>();
while (e.hasNext()){
    Token t = e.next();
    if (t.isOperation){
        Token op2=pending.pop();
        pending.push(t.eval(pending.pop(), op2));
    }else{
        pending.push(t);
    }
}
return pending.pop();
```

© 2008 Dr. Tim Margush

Infix Expressions

$$(2+3) * ((4-7/2)+9)$$

2 3 4 7 2 9

© 2008 Dr. Tim Margush

Infix Expression Parsing

- If **next** token is a number, output it
- If **next** token is an operation, check pending operations stack
 - While **top** has lower priority than **next**
 - pop and output
 - Push **next**

© 2008 Dr. Tim Margush

Infix Expressions

$$(2+3) * ((4-7/2)+9)$$

© 2008 Dr. Tim Margush

Associativity

- $2 + 3 - 5 + 1$
- $4 / 2 / 2$
- $2 ^ 3 ^ 2$
- $x = y = 4$

© 2008 Dr. Tim Margush

Base Conversions

- Given an int n, what is its representation in base b?
 - Division algorithm states $n = ab + r$ where r is the one's digit of the base b representation of n
 - Furthermore, the remaining digits are simply the base b representation of $a = n / b$
- This produces the digits in right to left order
- A Stack can be used to reverse them

© 2008 Dr. Tim Margush

Base Conversion

```
public static void display(int n, int base){
    Stack<Integer> digits = new Stack<Integer>();
    do{
        digits.push(n % base);
        n = n / base;
    } while(n>0);
    while (!digits.isEmpty())
        System.out.print(digits.pop());
    }
}
```

© 2008 Dr. Tim Margush

Matching Parenthesis

```
public static int mismatch(String s) { // examine "[{0({})][<>]" find mismatch
Stack<String> paren = new Stack<String>()
boolean OK = true;
int loc = -1;
while(OK && ++loc<s.length()){
if (isOpenParen(s.charAt(loc))
    paren.push(s.substring(loc,loc+1));
else
    if (parensMisMatch(s.charAt(loc),paren.pop())
        OK = false;
    }
if (loc==s.length()) loc=-1;
return loc;
}
```

© 2008 Dr. Tim Margush

Queue – Array Implementation

```
public class ArrayQueue<E>{
private Object [] elts = new Object[5];
private int front=0, rear=-1, size=0;
public E enqueue(E item){}
public E dequeue(){}
public E front(){}
boolean isFull(){}
boolean isEmpty(){}
...
}
```

© 2008 Dr. Tim Margush

Enqueue – Array Implementation

```
public class ArrayQueue<E>{
private Object [] elts = new Object[5];
private int front=0, rear=-1, size=0;
public E enqueue(E item){

}
}
```

© 2008 Dr. Tim Margush

Deque – Array Implementation

```
public class ArrayQueue<E>{  
    private Object [] elts = new Object[5];  
    private int front=0, rear=-1, size=0;  
    public E dequeue(){  
  
  
  
  
  
  
  
  
  
}
```

© 2008 Dr. Tim Margush

Queue – Linked List Implementation

```
public class LinkedQueue<E>{  
    private Node<E> front, rear;  
    {front = rear = new Node<E>();} //dummy header  
    public E enqueue(E item){  
  
  
  
  
  
  
  
  
  
}
```

© 2008 Dr. Tim Margush

Queue – Linked List Implementation

```
public class LinkedQueue<E>{  
    private Node<E> front, rear;  
    {front = rear = new Node<E>();}  
    public E dequeue() //makes 2nd node new dummy header  
  
  
  
  
  
  
  
  
  
}
```

© 2008 Dr. Tim Margush

Efficiency

- Both implementations are $O(1)$
- Array implementation may require resizing
- Array implementation implies wasted memory

© 2008 Dr. Tim Margush

Applications of Queues

- Operating system task scheduling
- Simulations – event lists
- Radix Sort
- Breadth First Search

© 2008 Dr. Tim Margush

DeQue

- Double-ended queue
 - DeQueue would be the proper abbreviation, but this is confused the dequeue operation
- Stack and queue combined in one structure
 - addFirst and addLast
 - removeFirst and removeLast

© 2008 Dr. Tim Margush

JFC

- Interface Queue
 - ArrayDeque, LinkedList, PriorityQueue
- Class Stack
 - Extends Vector (Array-based)
- Interface Deque
 - ArrayDeque, LinkedList

© 2008 Dr. Tim Margush

Summary

- Stacks and Queues are fundamental data structures supporting a variety of important applications
- Stack and Queue operations can be implemented to require $O(1)$ time
- Both array-based and linked implementations are plausible
- JFC provides good Stack and Queue classes

© 2008 Dr. Tim Margush
