Data Structures and Algorithms

Session 5. February 2, 2009

Instructor: Bert Huang

http://www.cs.columbia.edu/~bert/courses/3137

Announcements

- Homework 1 up on website
  - Due at beginning of next class

Review

- Dissected textbook's Java implementation of MyLinkedList and MyArrayList
  - add(), remove(), get(), set()
  - void vs. boolean add() – boolean is for Collections interface, which requires that add() returns whether the Collection changed

Today

- New Abstract Data Type: Stack
- Stack example applications
- Stack Implementations (easy)

Stack Definition

- Essentially a very restricted List
- Two (main) operations:
  - Push(AnyType x)
  - Pop(AnyType x)
- Analogy – Cafeteria Trays, PEZ

Stack Applications

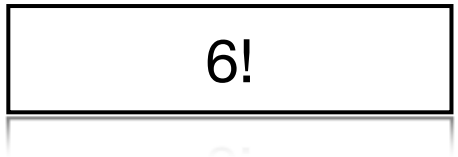
- Recursion
- Parsing text: infix vs. postfix
- Syntax checking ( ), { }, “”

Evaluating Recursion

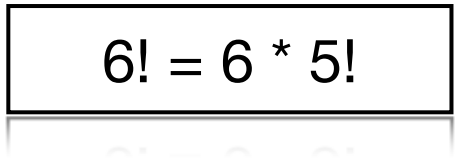
- Push recursive calls onto a Stack, evaluate top
- Consider computing factorials:
  - N! = N * (N-1)!
  - 1! = 1
- (Note: O(N!) is REALLY bad)

Stack Animation

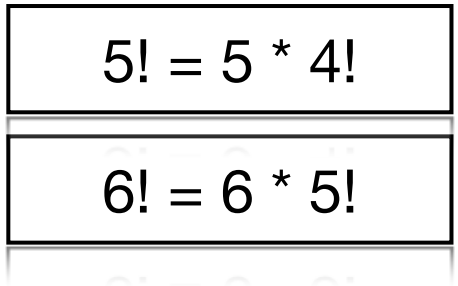
Stack Animation



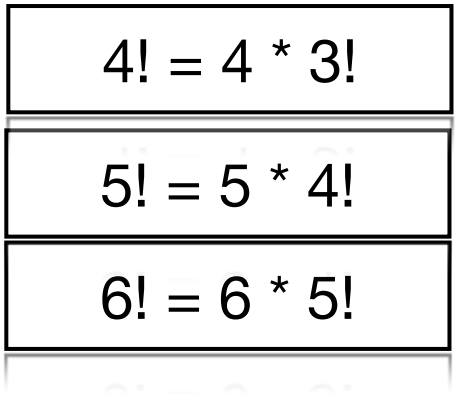
Stack Animation



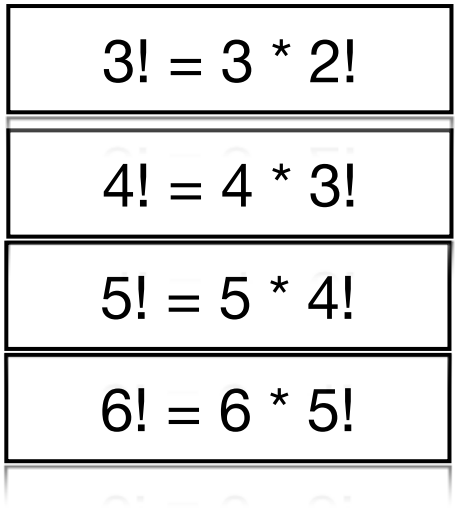
Stack Animation



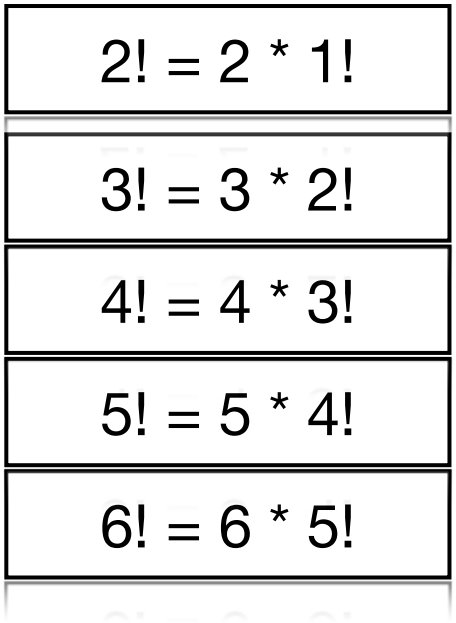
Stack Animation



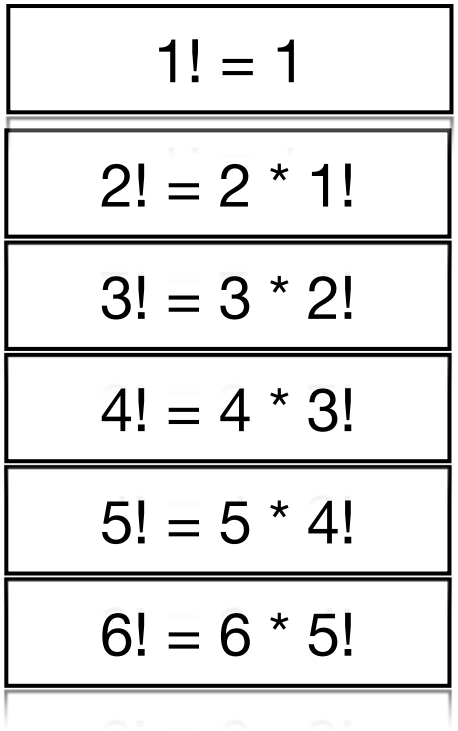
Stack Animation



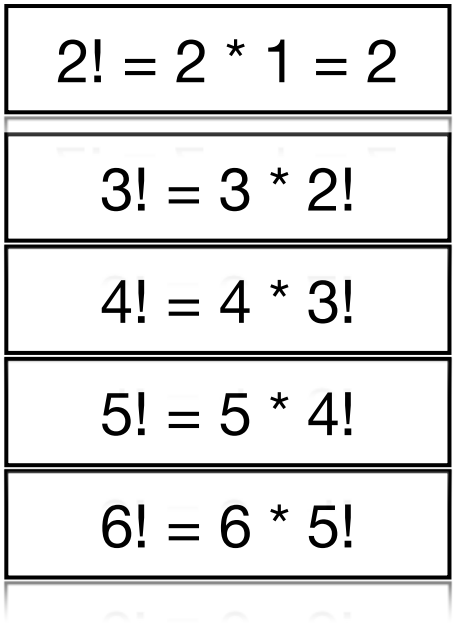
Stack Animation



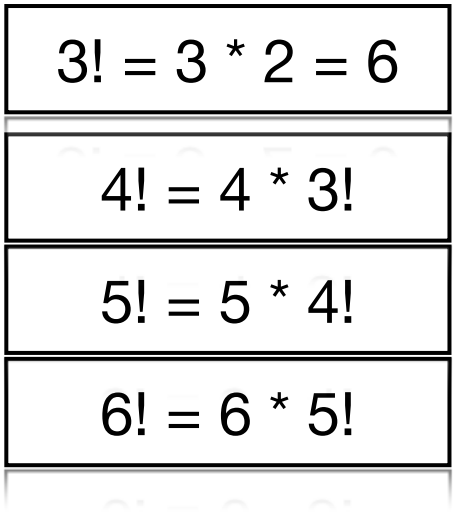
Stack Animation



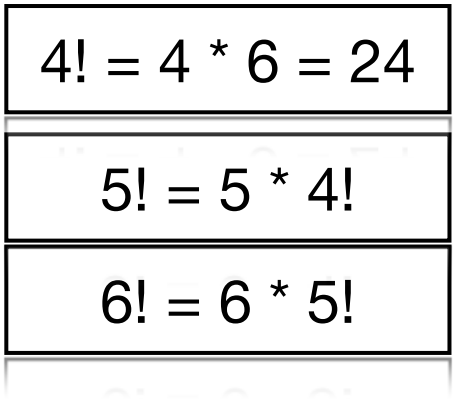
Stack Animation



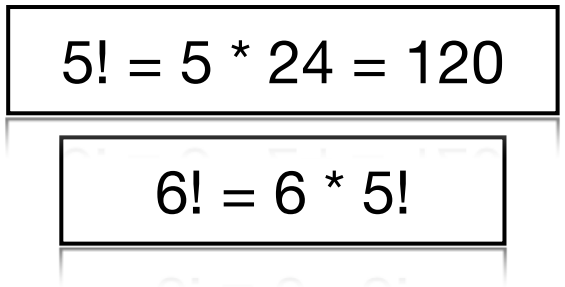
Stack Animation



Stack Animation



Stack Animation



Stack Animation

6! = 6 * 120 = 720

Evaluating Postfix

- Postfix notation places operator after operands
  - Ambiguous Infix: (3 + 2)* 10 ((3+2) * 10)
  - Postfix: 3 2 + 10 * ((3 2 +) 10 *)

(As opposed to) 3 2 10 * + (3 (2 10 *) +)

Evaluating Postfix

- Postfix notation places operator after operands
  - Ambiguous Infix: 3 + 2 * 10 ((3+2) * 10)
  - Postfix: 3 2 + 10 * ((3 2 +) 10 *)

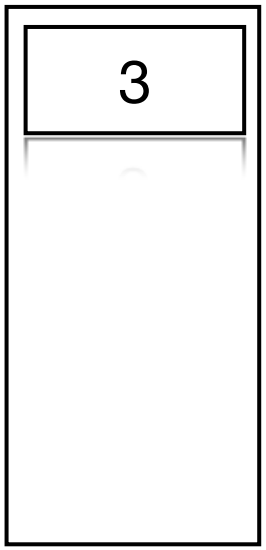
(As opposed to) 3 2 10 * + (3 (2 10 *) +)

Postfix Stack

- Push symbols as they appear
- Whenever we read an operator, pop two operands
  - Evaluate operation, push result
- E.g., 3 2 + 10 *

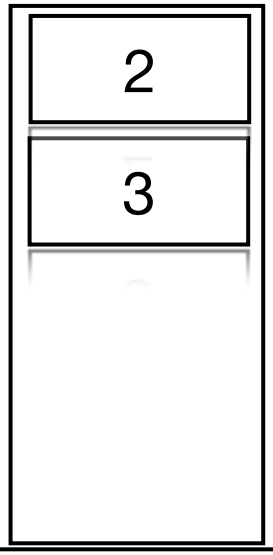
Postfix Stack

- Push symbols as they appear
- Whenever we read an operator, pop two operands
  - Evaluate operation, push result
- E.g., 3 2 + 10 *



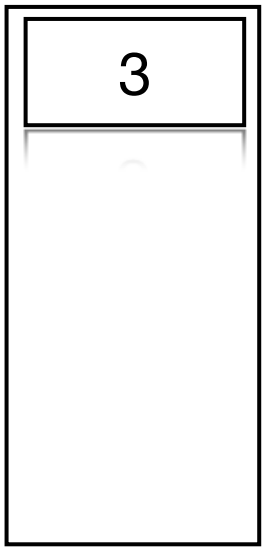
Postfix Stack

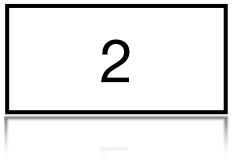
- Push symbols as they appear
- Whenever we read an operator, pop two operands
  - Evaluate operation, push result
- E.g., 3 2 + 10 *



Postfix Stack

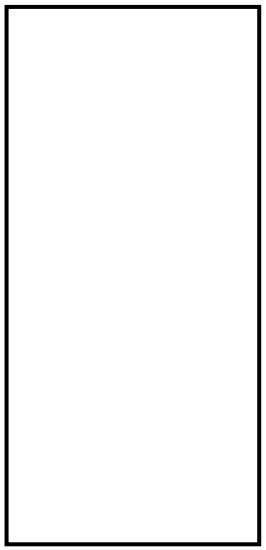
- Push symbols as they appear
- Whenever we read an operator, pop two operands
  - Evaluate operation, push result
- E.g., 3 2 + 10 *





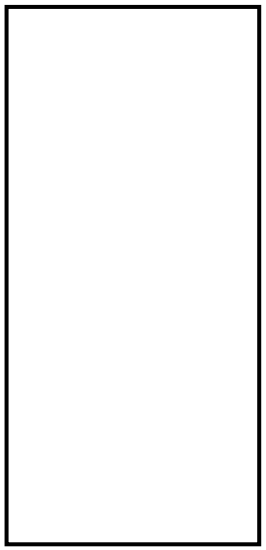
Postfix Stack

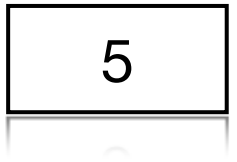
- Push symbols as they appear
- Whenever we read an operator, pop two operands
  - Evaluate operation, push result
- E.g., 3 2 + 10 *



Postfix Stack

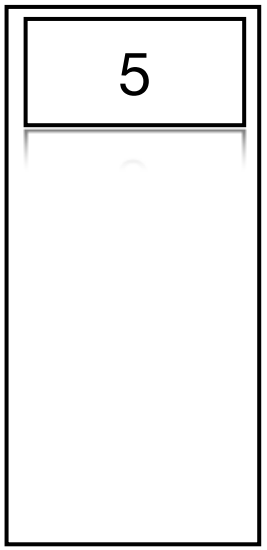
- Push symbols as they appear
- Whenever we read an operator, pop two operands
  - Evaluate operation, push result
- E.g., 3 2 + 10 *





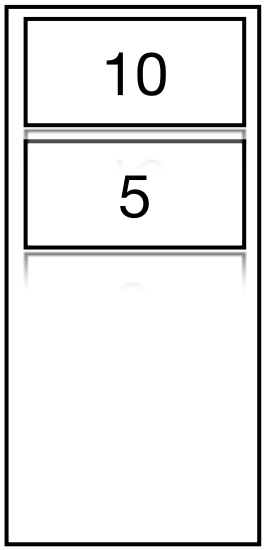
Postfix Stack

- Push symbols as they appear
- Whenever we read an operator, pop two operands
  - Evaluate operation, push result
- E.g., 3 2 + 10 *



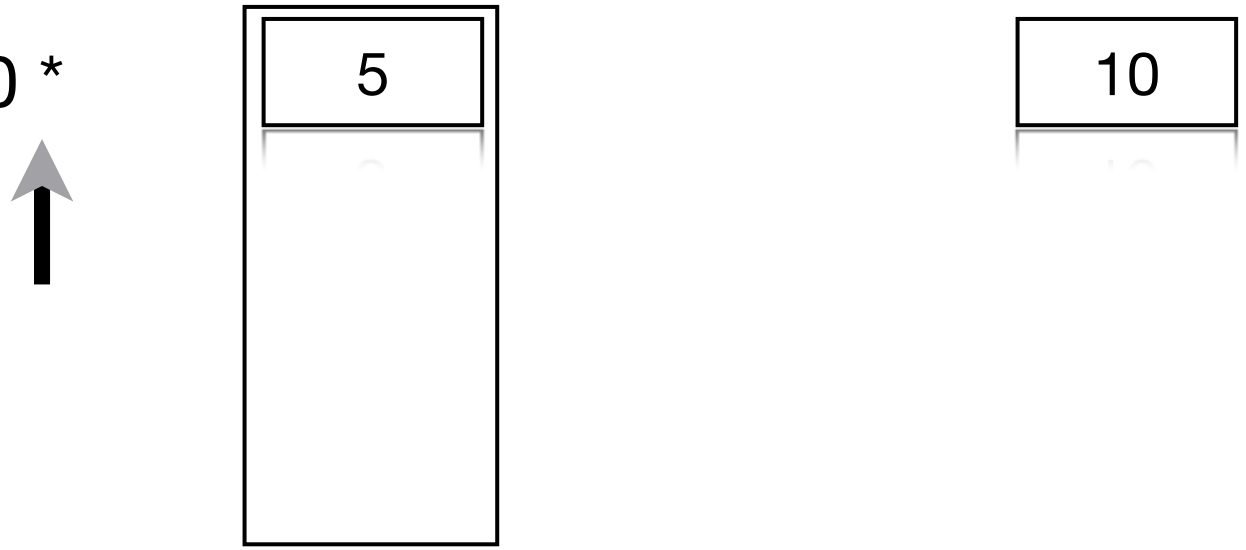
Postfix Stack

- Push symbols as they appear
- Whenever we read an operator, pop two operands
  - Evaluate operation, push result
- E.g., 3 2 + 10 *



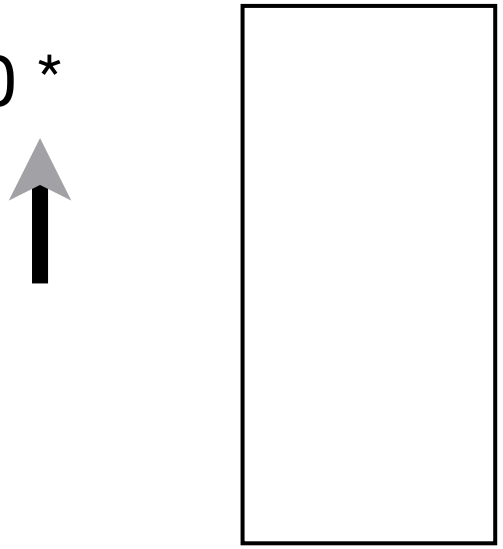
Postfix Stack

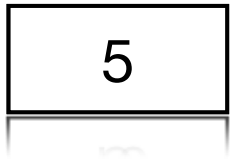
- Push symbols as they appear
- Whenever we read an operator, pop two operands
  - Evaluate operation, push result
- E.g., 3 2 + 10 *



Postfix Stack

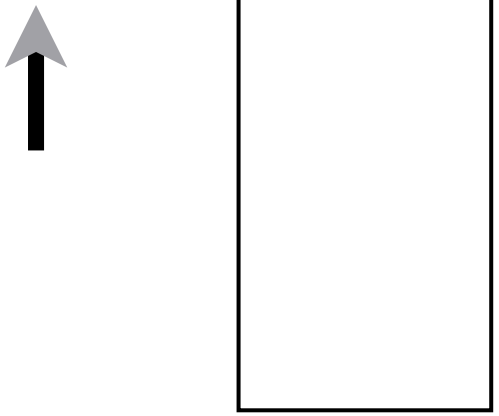
- Push symbols as they appear
- Whenever we read an operator, pop two operands
  - Evaluate operation, push result
- E.g., 3 2 + 10 *

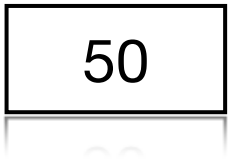




Postfix Stack

- Push symbols as they appear
- Whenever we read an operator, pop two operands
  - Evaluate operation, push result
- E.g., 3 2 + 10 *





Syntax Checking

- Check for matching parenthesis ( ), braces { }, brackets [ ], etc.
- Sweep through code
  - If we see an opening symbol, push onto stack
  - If we see a closing symbol, pop from stack and compare

Syntax Checking

```
public void add( int idx, AnyType x)
{ if( theItems.length == size( ) ) ensureCapacity( size
( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
theSize++; }
```

Syntax Checking

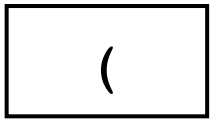
```
public void add( int idx, AnyType x)
{ if( theItems.length == size( ) ) ensureCapacity( size
( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
theSize++; }
```

Syntax Checking

```
public void add( int idx, AnyType x)
{ if( theItems.length == size( ) ) ensureCapacity( size
( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
theSize++; }
```

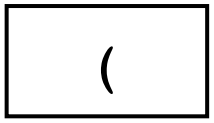
Syntax Checking

```
public void add( int idx, AnyType x)
{ if( theItems.length == size( ) ) ensureCapacity( size
( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
theSize++; }
```



Syntax Checking

- ```
  public void add( int idx, AnyType x)
  { if( theItems.length == size( ) ) ensureCapacity( size
  ( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
  theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
  theSize++; }
  ```


  ```

Syntax Checking

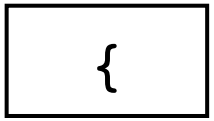
```
public void add( int idx, AnyType x)
{ if( theItems.length == size( ) ) ensureCapacity( size
( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
theSize++; }
```

Syntax Checking

```
public void add( int idx, AnyType x)
{ if( theItems.length == size( ) ) ensureCapacity( size
( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
theSize++; }
```

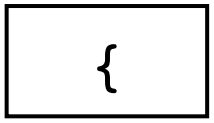
Syntax Checking

```
public void add( int idx, AnyType x)
{ if( theItems.length == size( ) ) ensureCapacity( size
( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
theSize++; }
```



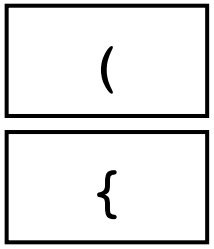
Syntax Checking

```
public void add( int idx, AnyType x)
{ if( theItems.length == size( ) ) ensureCapacity( size
( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
theSize++; }
```



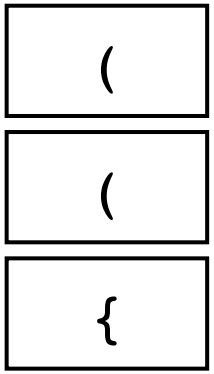
Syntax Checking

- ```
  public void add( int idx, AnyType x)
  { if( theItems.length == size( ) ) ensureCapacity( size
  ( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
  theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
  theSize++; }
  ```


  ```

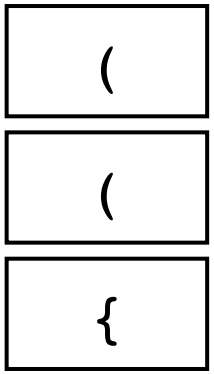
Syntax Checking

```
public void add( int idx, AnyType x)
{ if( theItems.length == size( ) ) ensureCapacity( size
( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
theSize++; }
```



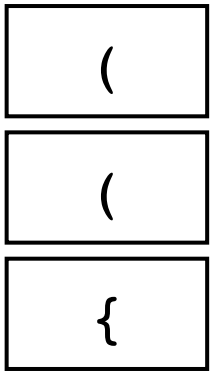
Syntax Checking

```
public void add( int idx, AnyType x)
{ if( theItems.length == size( ) ) ensureCapacity( size
( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
theSize++; }
```



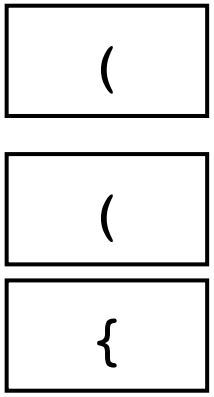
Syntax Checking

```
public void add( int idx, AnyType x)
{ if( theItems.length == size( ) ) ensureCapacity( size
( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
theSize++; }
```



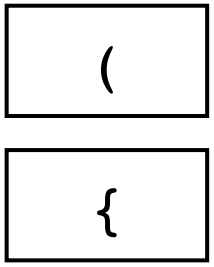
Syntax Checking

```
public void add( int idx, AnyType x)
{ if( theItems.length == size( ) ) ensureCapacity( size
( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
theSize++; }
```



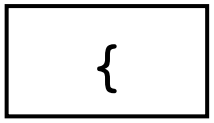
Syntax Checking

```
public void add( int idx, AnyType x)
{ if( theItems.length == size( ) ) ensureCapacity( size
( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
theSize++; }
```



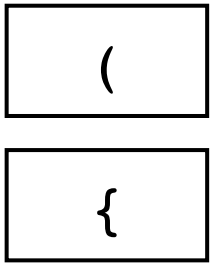
Syntax Checking

```
public void add( int idx, AnyType x)
{ if( theItems.length == size( ) ) ensureCapacity( size
( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
theSize++; }
```



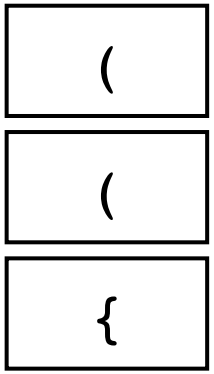
Syntax Checking

```
public void add( int idx, AnyType x)
{ if( theItems.length == size( ) ) ensureCapacity( size
( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
theSize++; }
```



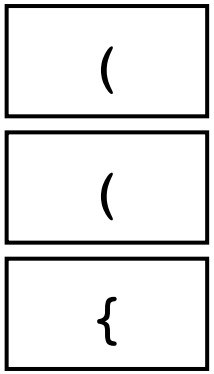
Syntax Checking

```
public void add( int idx, AnyType x)
{ if( theItems.length == size( ) ) ensureCapacity( size
( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
theSize++; }
```



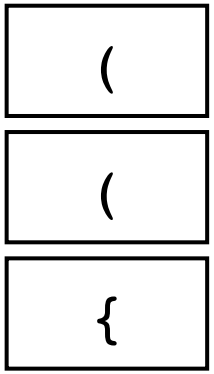
Syntax Checking

```
public void add( int idx, AnyType x)
{ if( theItems.length == size( ) ) ensureCapacity( size
( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
theSize++; }
```



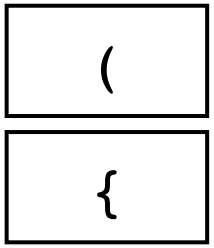
Syntax Checking

- ```
  public void add( int idx, AnyType x)
  { if( theItems.length == size( ) ) ensureCapacity( size
  ( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
  theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
  theSize++; }
  ```


  ```

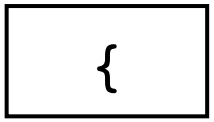
Syntax Checking

```
public void add( int idx, AnyType x)
{ if( theItems.length == size( ) ) ensureCapacity( size
( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
theSize++; }
```



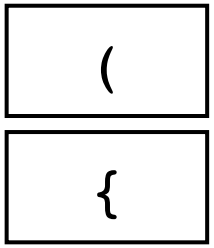
Syntax Checking

```
public void add( int idx, AnyType x)
{ if( theItems.length == size( ) ) ensureCapacity( size
( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
theSize++; }
```



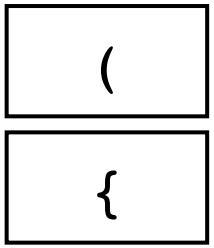
Syntax Checking

```
public void add( int idx, AnyType x)
{ if( theItems.length == size( ) ) ensureCapacity( size
( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
theSize++; }
```



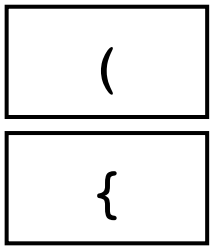
Syntax Checking

```
public void add( int idx, AnyType x)
{ if( theItems.length == size( ) ) ensureCapacity( size
( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
theSize++; }
```



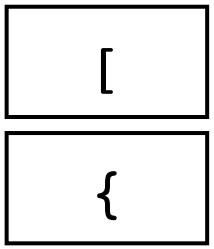
Syntax Checking

```
public void add( int idx, AnyType x)
{ if( theItems.length == size( ) ) ensureCapacity( size
( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
theSize++; }
```



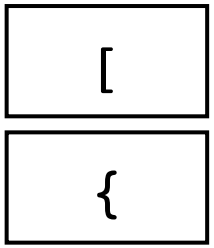
Syntax Checking

```
public void add( int idx, AnyType x)
{ if( theItems.length == size( ) ) ensureCapacity( size
( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
theSize++; }
```



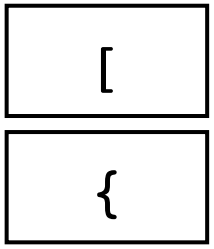
Syntax Checking

```
public void add( int idx, AnyType x)
{ if( theItems.length == size( ) ) ensureCapacity( size
( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
theSize++; }
```



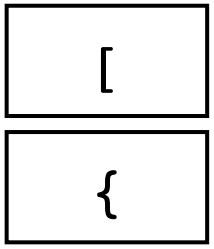
Syntax Checking

```
public void add( int idx, AnyType x)
{ if( theItems.length == size( ) ) ensureCapacity( size
( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
theSize++; }
```



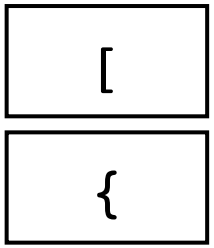
Syntax Checking

```
public void add( int idx, AnyType x)
{ if( theItems.length == size( ) ) ensureCapacity( size
( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
theSize++; }
```



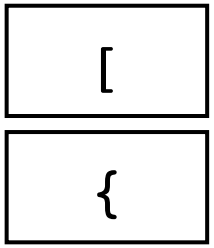
Syntax Checking

```
public void add( int idx, AnyType x)
{ if( theItems.length == size( ) ) ensureCapacity( size
( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
theSize++; }
```



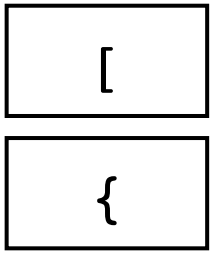
Syntax Checking

```
public void add( int idx, AnyType x)
{ if( theItems.length == size( ) ) ensureCapacity( size
( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
theSize++; }
```



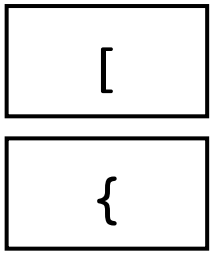
Syntax Checking

```
public void add( int idx, AnyType x)
{ if( theItems.length == size( ) ) ensureCapacity( size
( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
theSize++; }
```



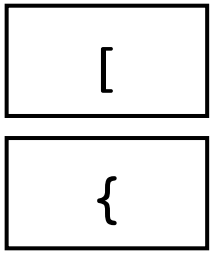
Syntax Checking

```
public void add( int idx, AnyType x)
{ if( theItems.length == size( ) ) ensureCapacity( size
( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
theSize++; }
```



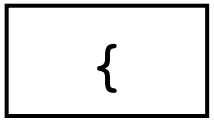
Syntax Checking

```
public void add( int idx, AnyType x)
{ if( theItems.length == size( ) ) ensureCapacity( size
( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
theSize++; }
```



Syntax Checking

```
public void add( int idx, AnyType x)
{ if( theItems.length == size( ) ) ensureCapacity( size
( ) * 2 + 1); for( int i=theSize; i > idx; i-- )
theItems[ i ] = theItems[ i - 1 ]; theItems[ idx ] = x;
theSize++; }
```



Stack Implementations

- Linked List:
  - Push(x) <-> add(x) <-> add(x,0)
  - Pop(x) <-> remove(0)
- Array:
  - Push(x) <-> Array[k++] = x
  - Pop(x) <-> return Array[--k]

Assignments

- Homework 1 due next class
- Homework 2 will be assigned next class, will cover Lists, Stacks, Queues
- Weiss Section 3.7 – Queues