Data Structures in Java

Session 6

Instructor: Bert Huang

http://www1.cs.columbia.edu/~bert/courses/3134

Announcements

- Homework 2 released on website
  - Due Oct. 6th at 5:40 PM (12 days)

Review

- List code study
  - MyArrayList
  - MyLinkedList
- The Iterable interface
- Definition of Stack ADT

Today's Plan

- Homework advice
- Review of **scope** and recursion
- Stack applications examples
- Stack implementation (easy)

Test First

- Incrementally test your code
  - avoid having to debug too many moving parts at once
- 1st step, make it compile (have methods return garbage)
- 2nd step, make methods pass tests

Scope

-
  ```
  public int x = 200;

  public void mystery() {
      int x = 100;
      System.out.println(x);
  }
  ```

-
  ```
  int i = 200;
  for (int i=0; i<10; i++)
      System.out.println(“i = “+i);
  System.out.println(“i = “+i);
  ```

Generics and Scope

-
  ```
  public class Collection<String> {
      String whatIsThis;
  }
  Collection<Integer> myCollection =
        new Collection<Integer>();
  ```
- whatIsThis is an Integer
- So be careful with naming your generic placeholders

Stack Definition

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

Stack Applications

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

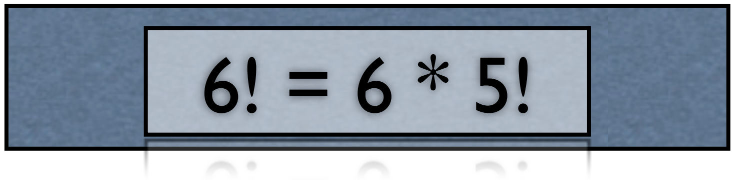
Evaluating Recursion

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

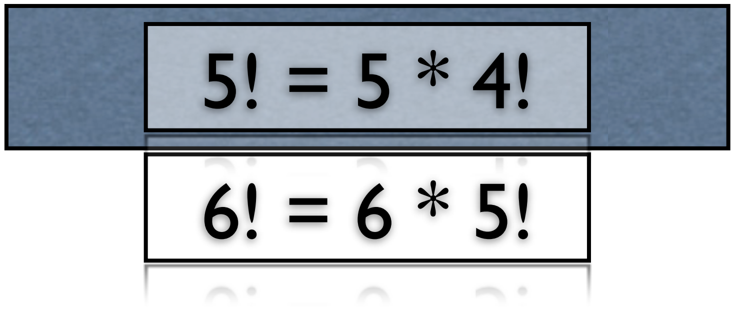
Stack Animation

Stack Animation

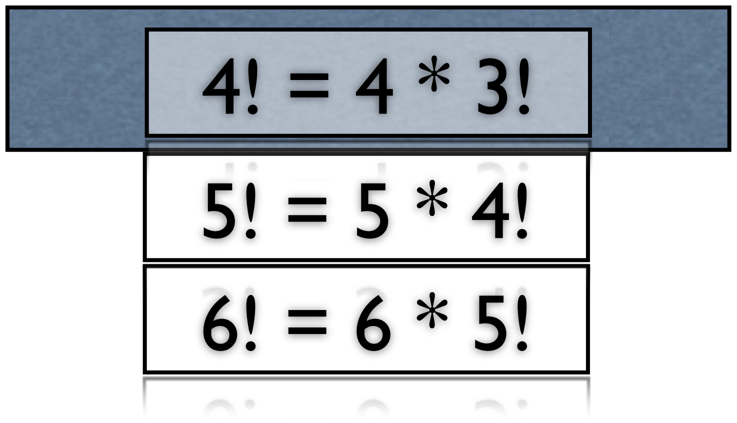
Stack Animation



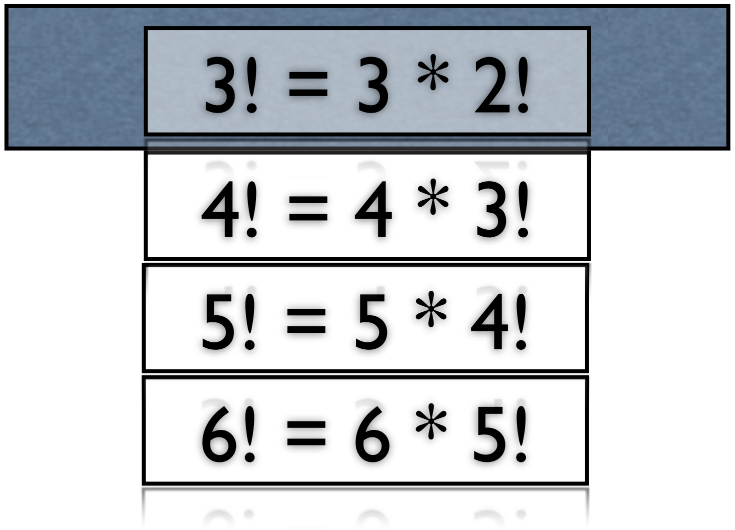
Stack Animation



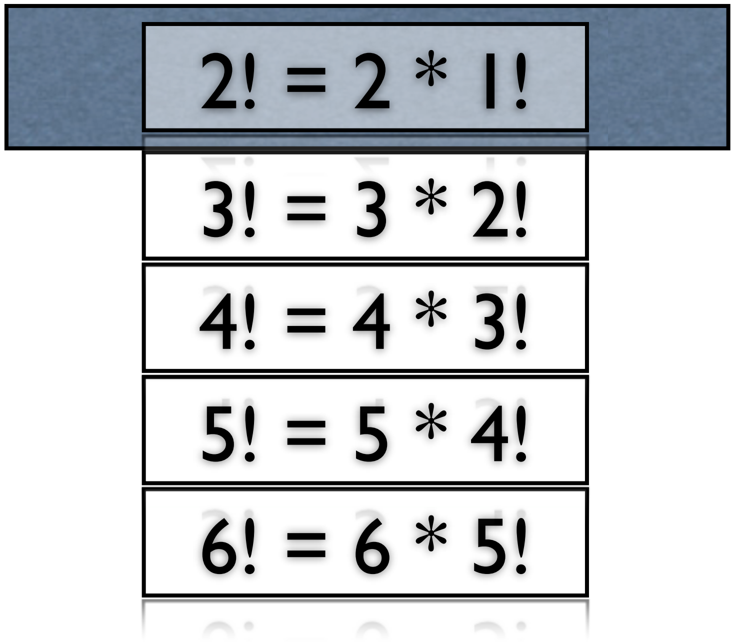
Stack Animation



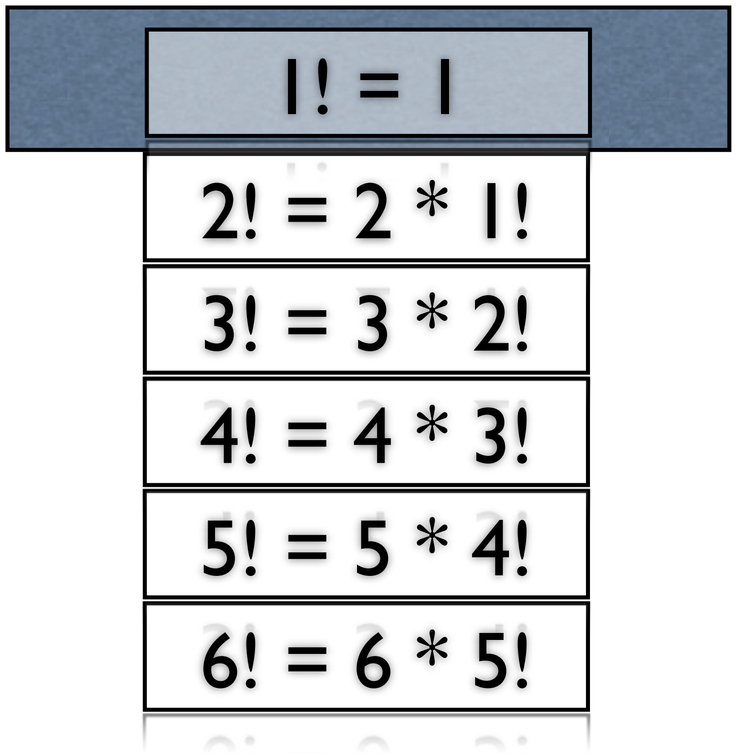
Stack Animation



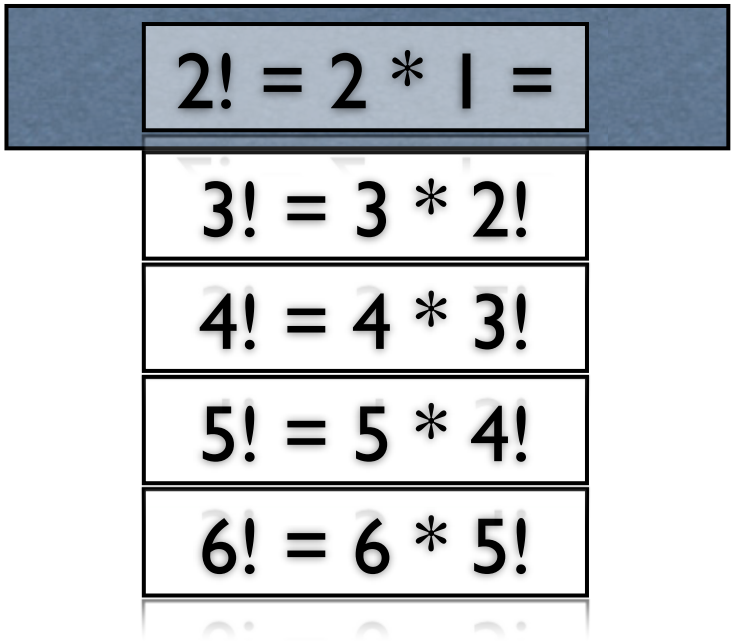
Stack Animation



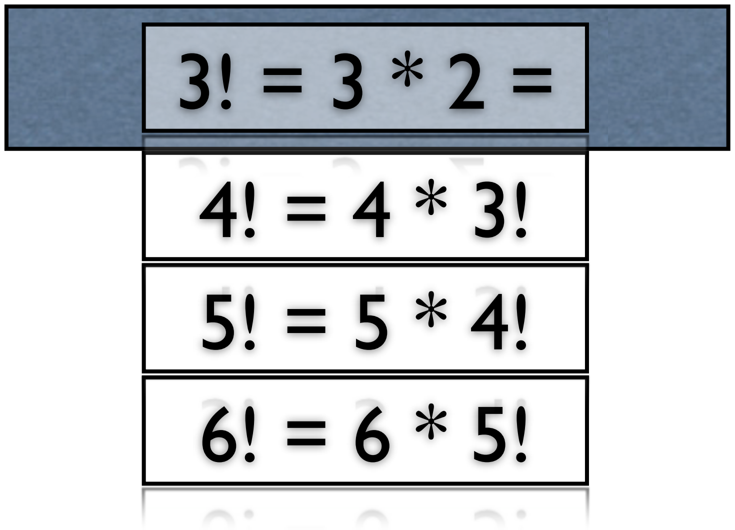
Stack Animation



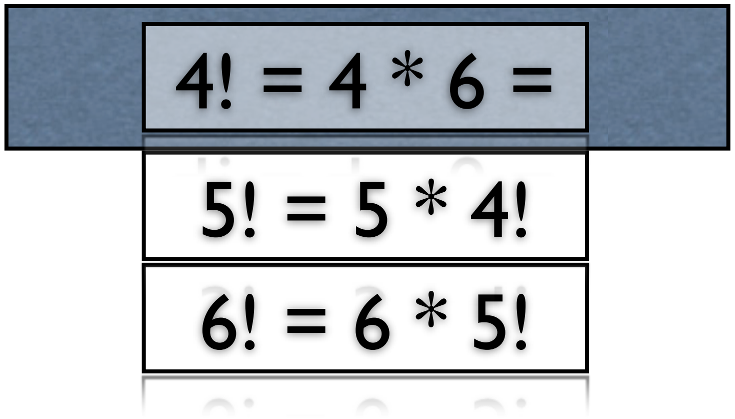
Stack Animation



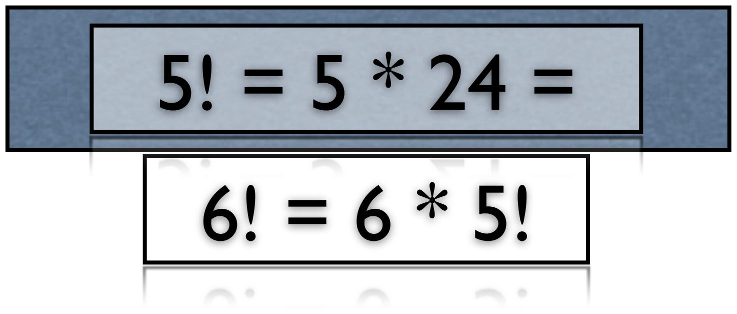
Stack Animation



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 *) +)

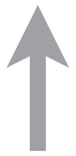
Evaluating Postfix

- 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 *) +)

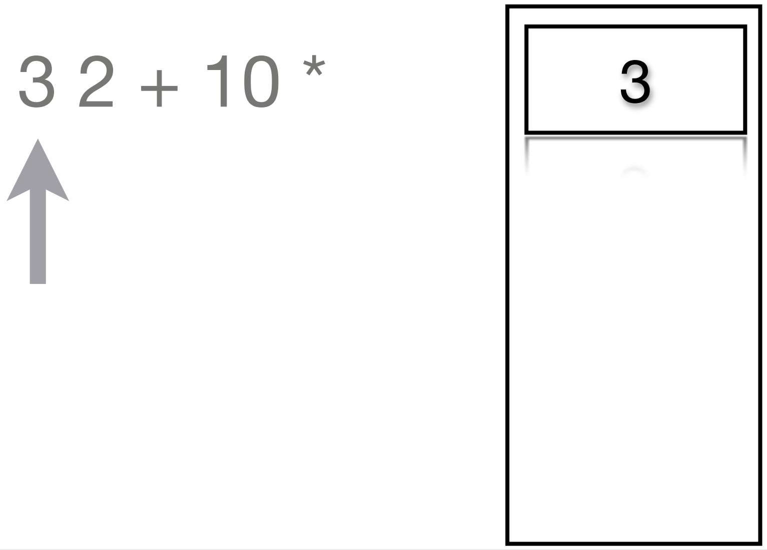
Postfix Stack

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



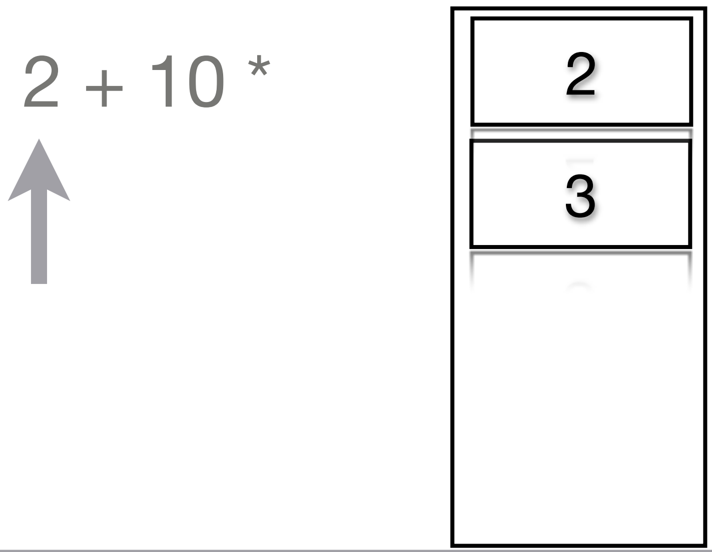
Postfix Stack

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



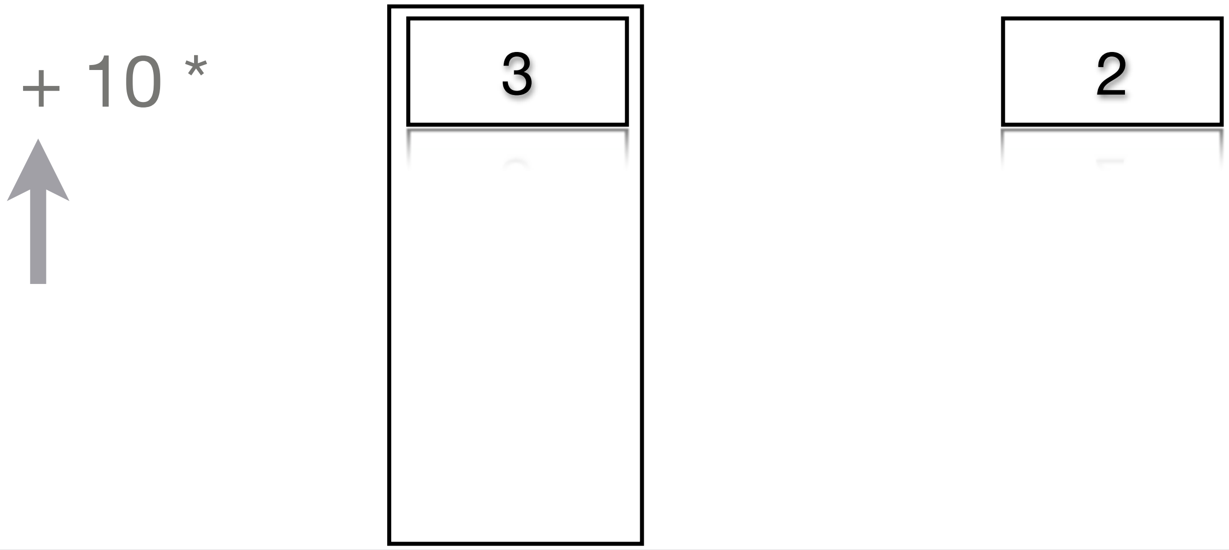
Postfix Stack

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



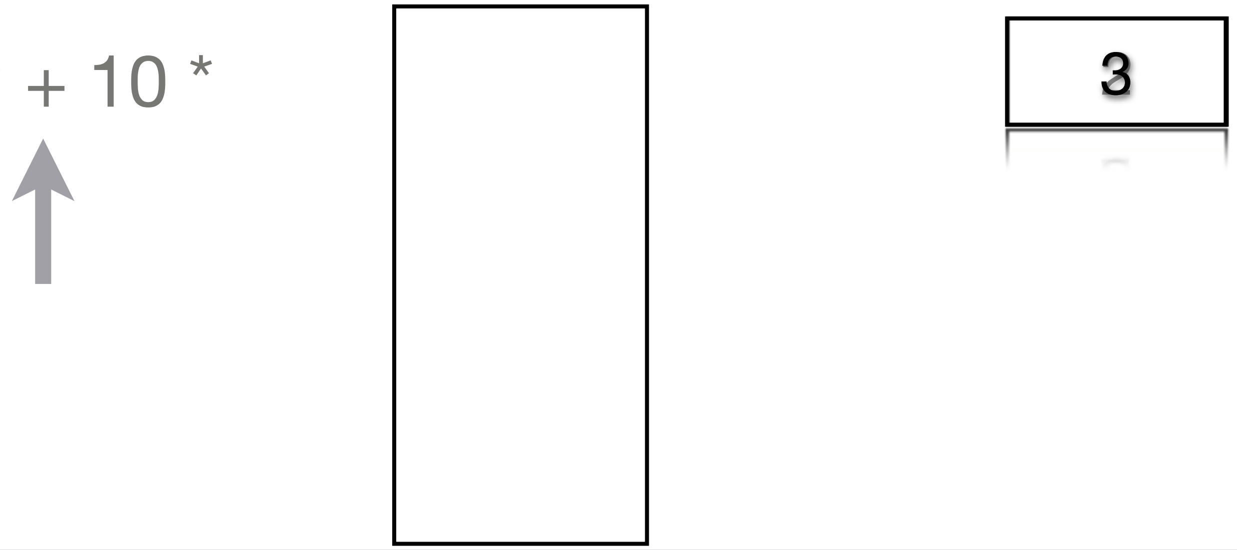
Postfix Stack

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



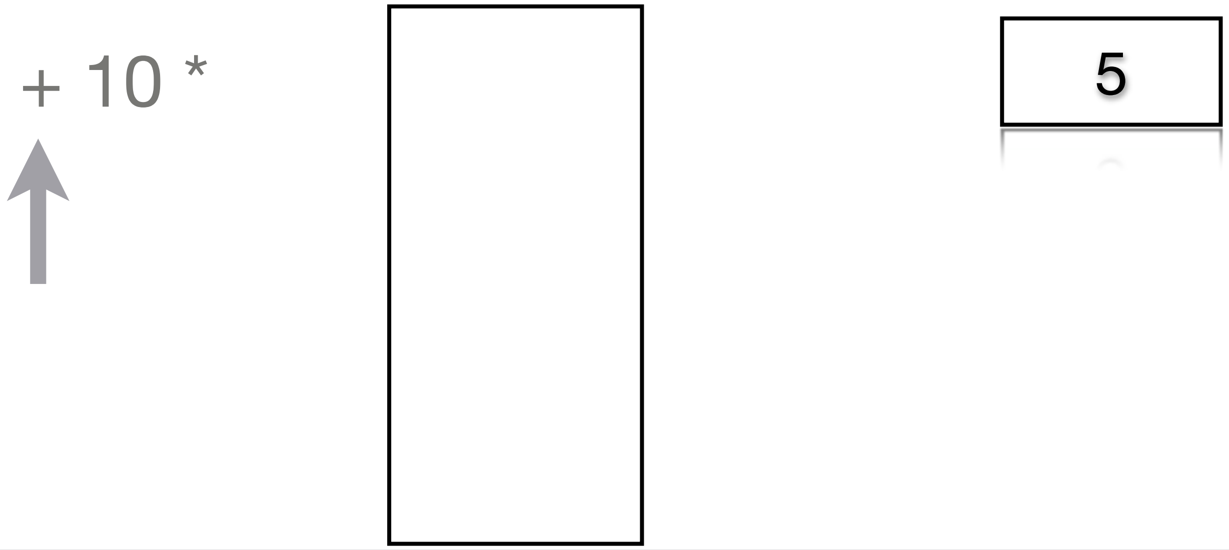
Postfix Stack

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



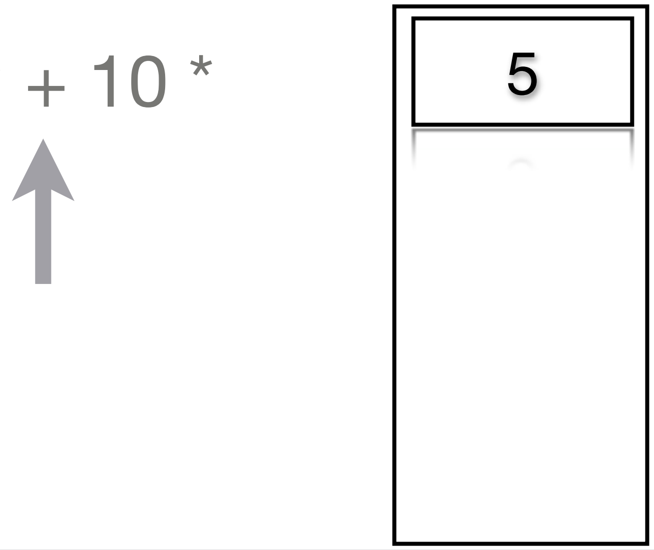
Postfix Stack

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



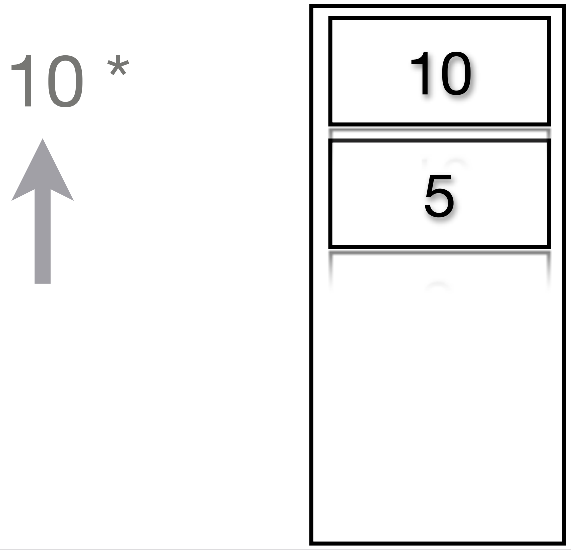
Postfix Stack

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



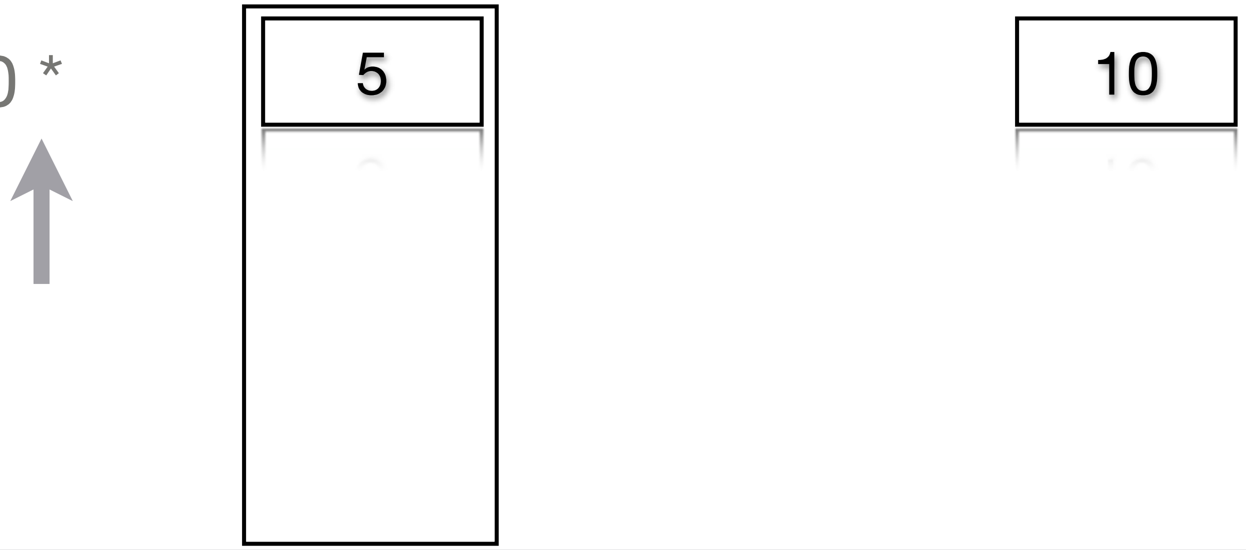
Postfix Stack

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



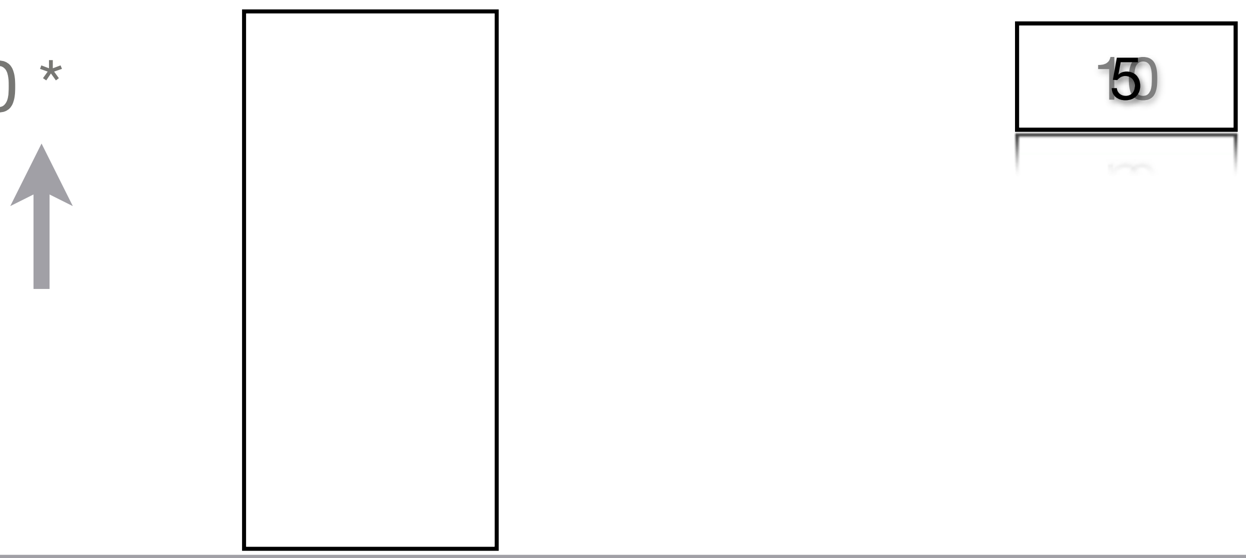
Postfix Stack

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



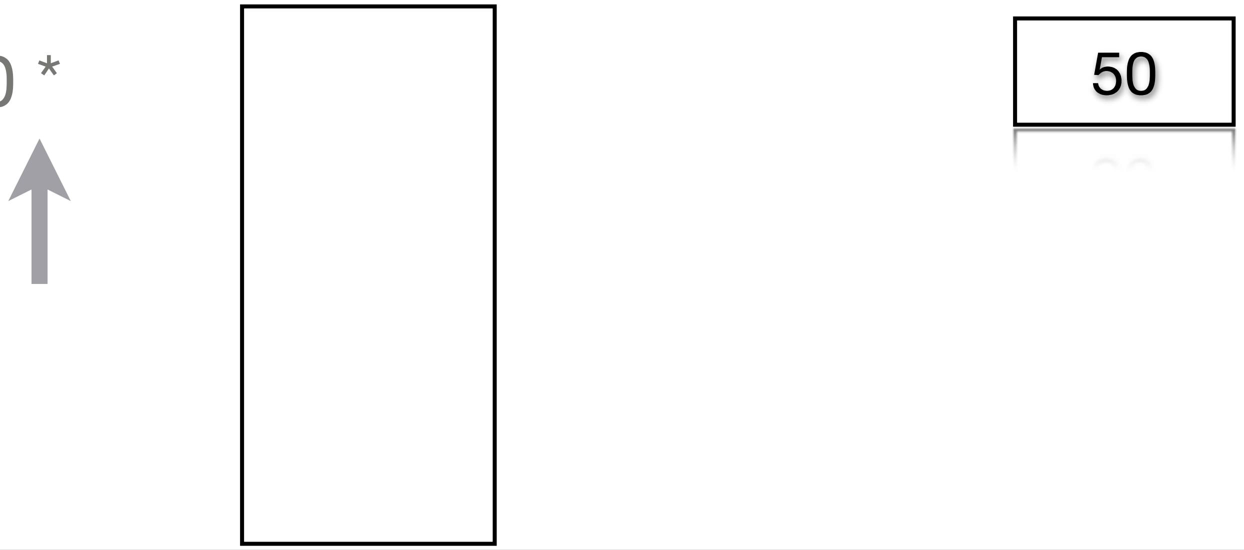
Postfix Stack

- 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

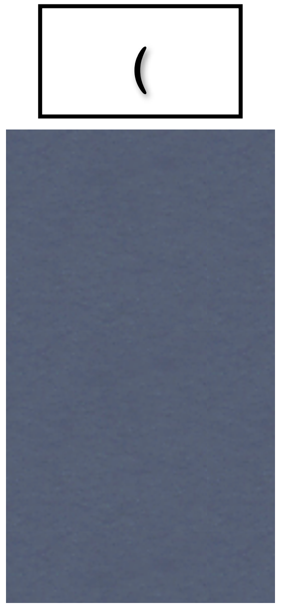
-
  ```
  public void add( int idx, AnyType x) { if
  ( theItems.length == size( ) ) ensureCapacity( size ( )
  * 2 + 1); for( int i=theSize; i > idx; i-- ) theItems
  [ i ] = theItems[ i - 1 ]; theItems[ idx ] = x; theSize+
  +; }
  ```

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++; }`

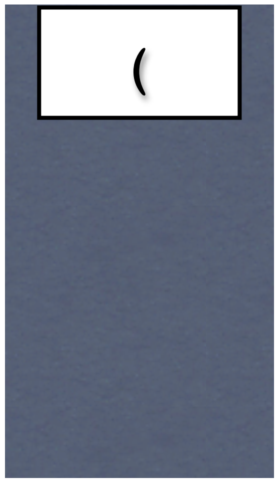
Syntax Checking

* `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++; }`



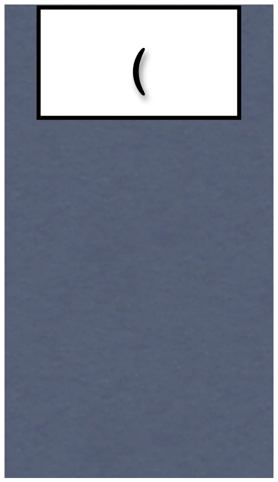
Syntax Checking

- `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++; }

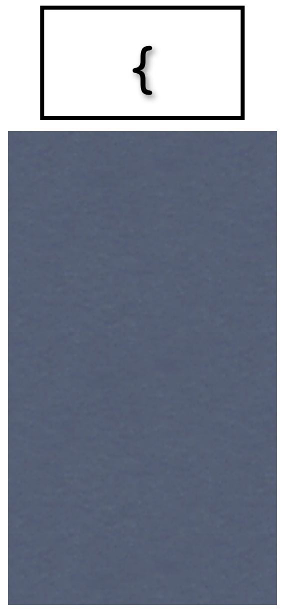


Syntax Checking

- `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++; }`

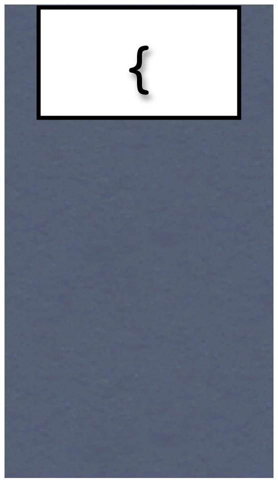
Syntax Checking

- `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++; }`



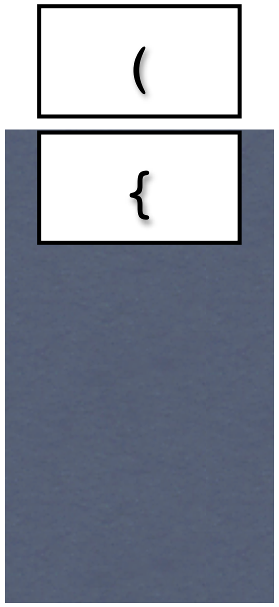
Syntax Checking

- `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++; }`



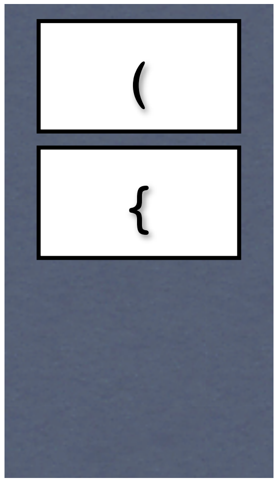
Syntax Checking

-
  ```
  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+
  +; }
  ```



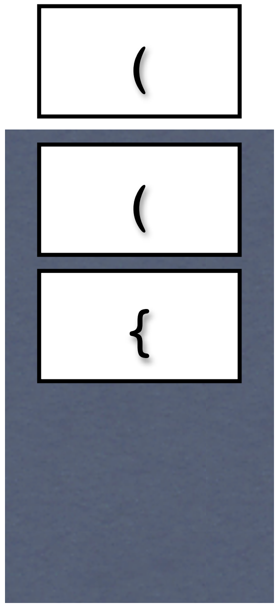
Syntax Checking

- `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++; }`



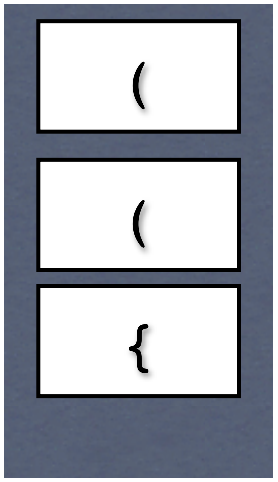
Syntax Checking

- `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++; }`



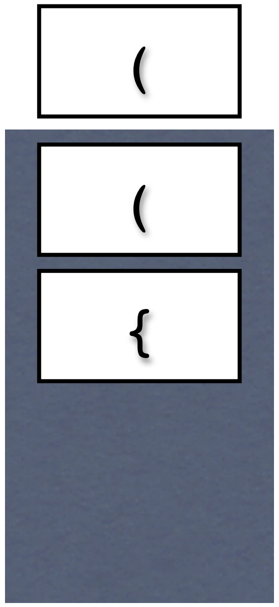
Syntax Checking

- `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++; }`



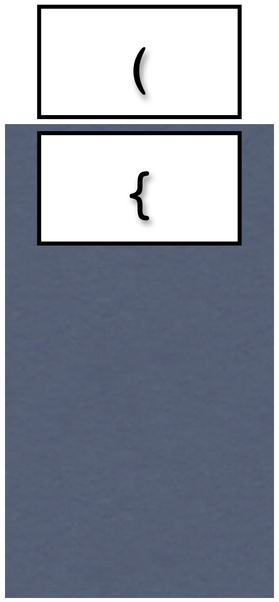
Syntax Checking

- `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++; }`



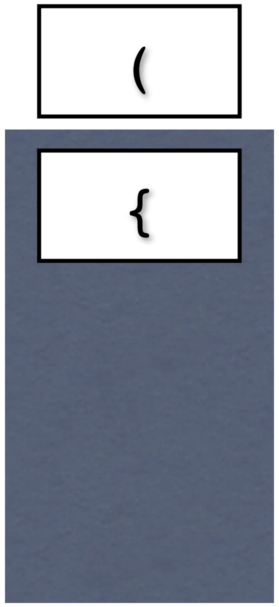
Syntax Checking

- `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++; }`



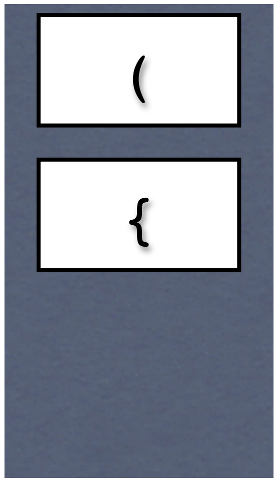
Syntax Checking

- 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+ +; }



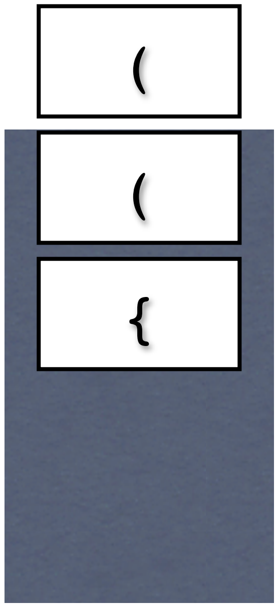
Syntax Checking

- 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++; }



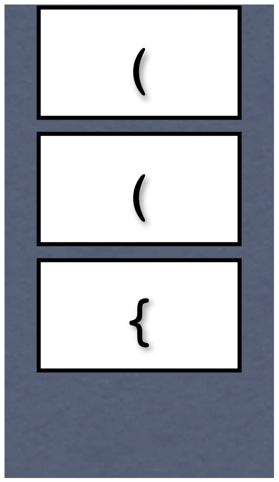
Syntax Checking

- `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+ +; }`



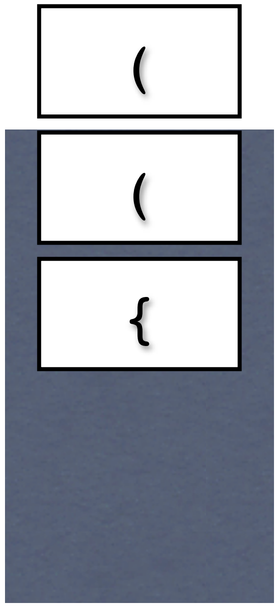
Syntax Checking

- `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++; }`



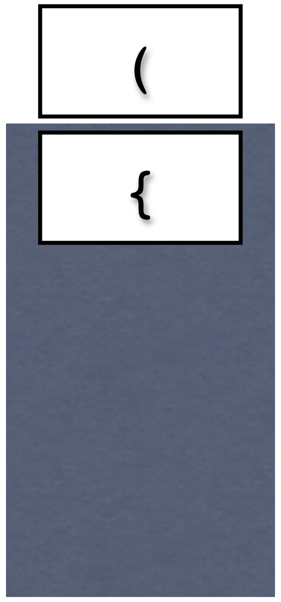
Syntax Checking

- `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++; }`



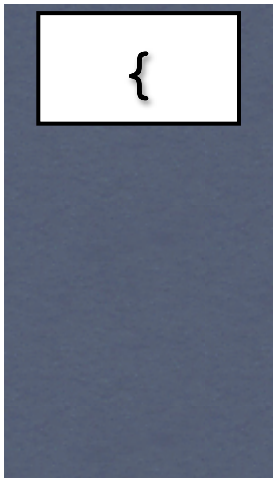
Syntax Checking

- 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++; }



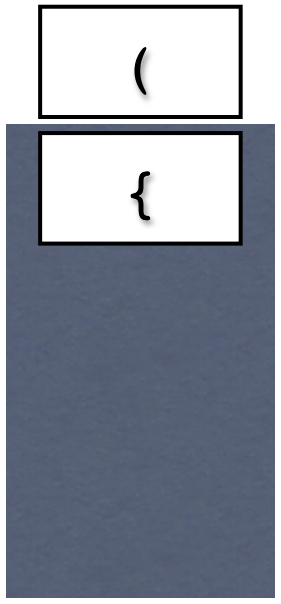
Syntax Checking

- `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++; }`



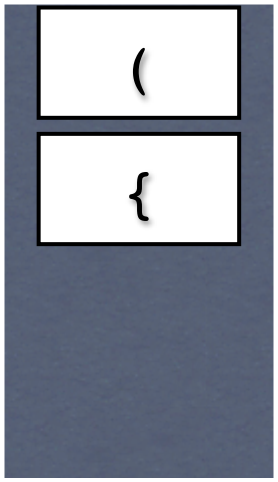
Syntax Checking

- `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++; }`



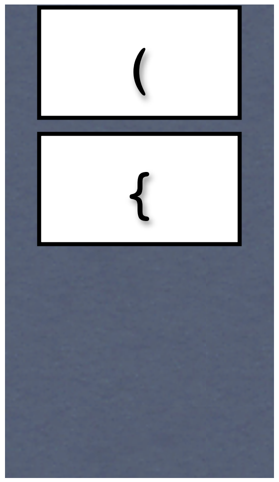
Syntax Checking

* `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++; }`



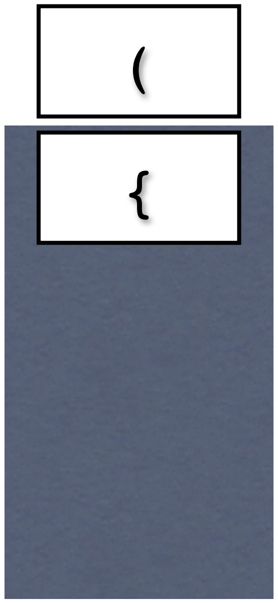
Syntax Checking

- `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+ +; }`



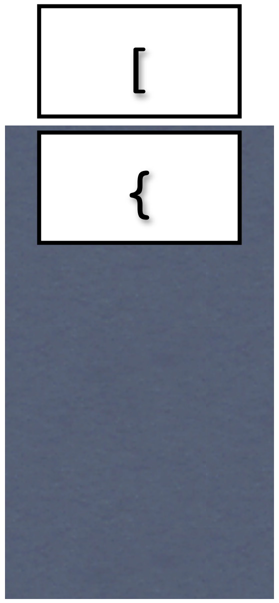
Syntax Checking

- `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++; }`



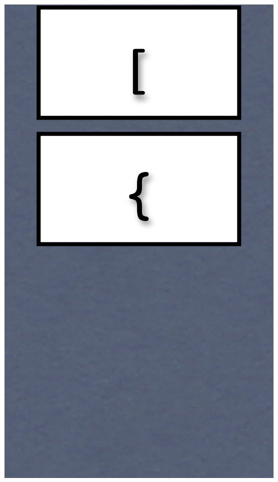
Syntax Checking

- `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++; }`



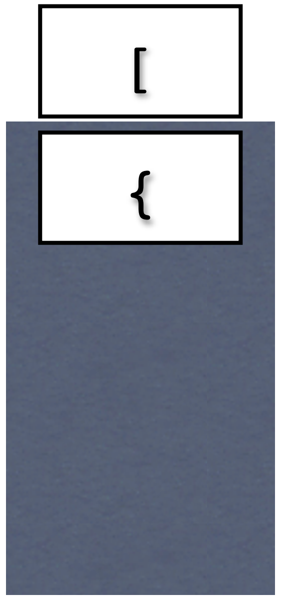
Syntax Checking

- `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++; }`



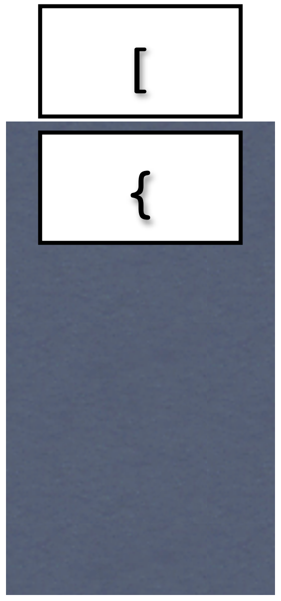
Syntax Checking

- `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++; }`



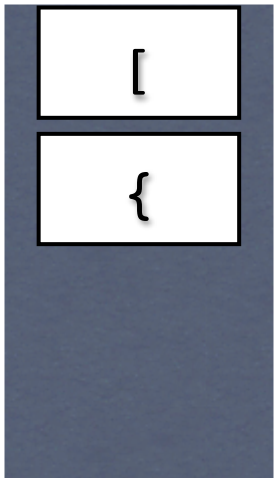
Syntax Checking

- `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++; }`



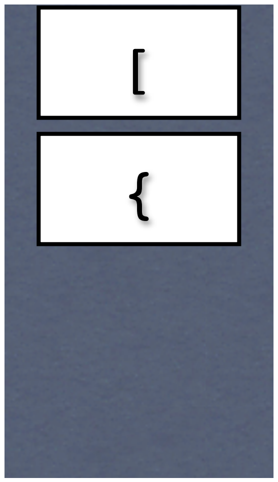
Syntax Checking

- `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++; }`



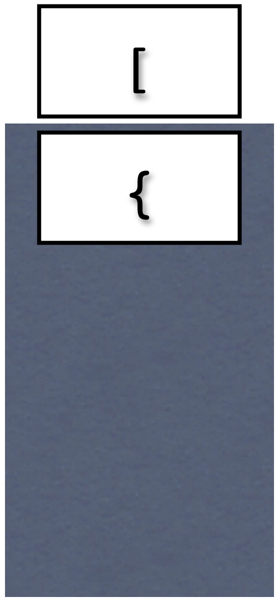
Syntax Checking

- `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++; }`



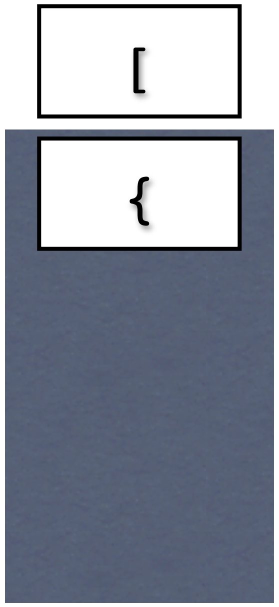
Syntax Checking

- `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++; }`



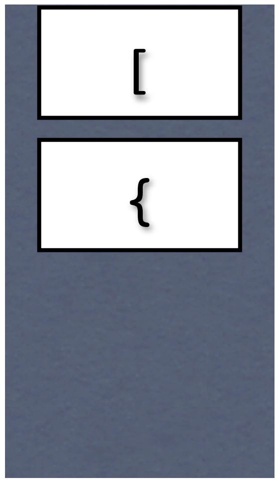
Syntax Checking

- `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++; }`



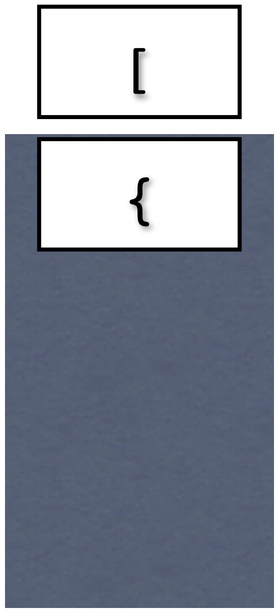
Syntax Checking

- `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++; }`



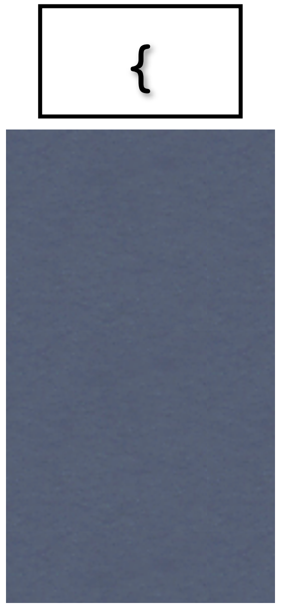
Syntax Checking

- `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++; }`



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

Queue ADT

- Stacks are **L**ast **I**n **F**irst **O**ut
- Queues are **F**irst **I**n **F**irst **O**ut, first-come first-served
- Operations: **enqueue** and **dequeue**
- Analogy: standing in line, garden hose, etc

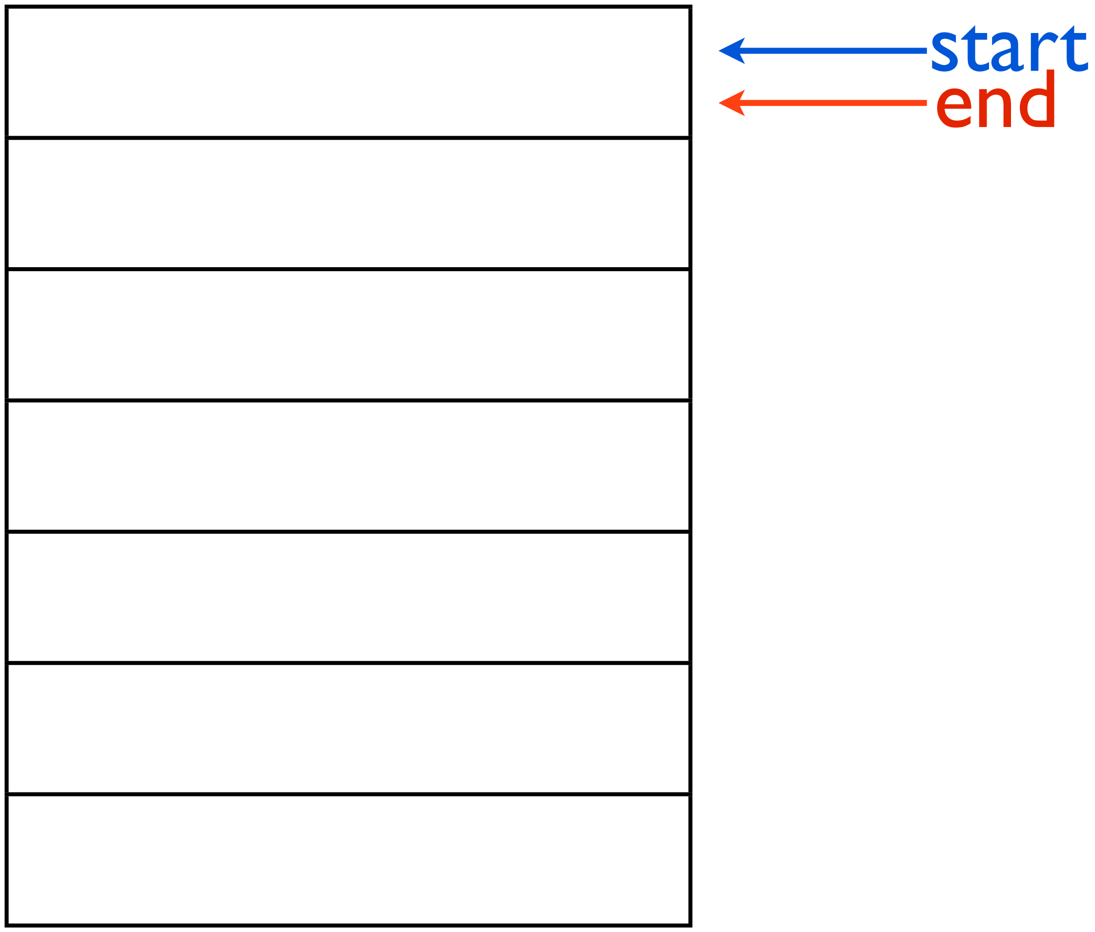
Queue Implementation

- Linked List
  - add(x,0) to enqueue, remove(N-1) to dequeue
- Array List won’t work well!
  - add(x,0) is expensive
  - Solution: use a circular array

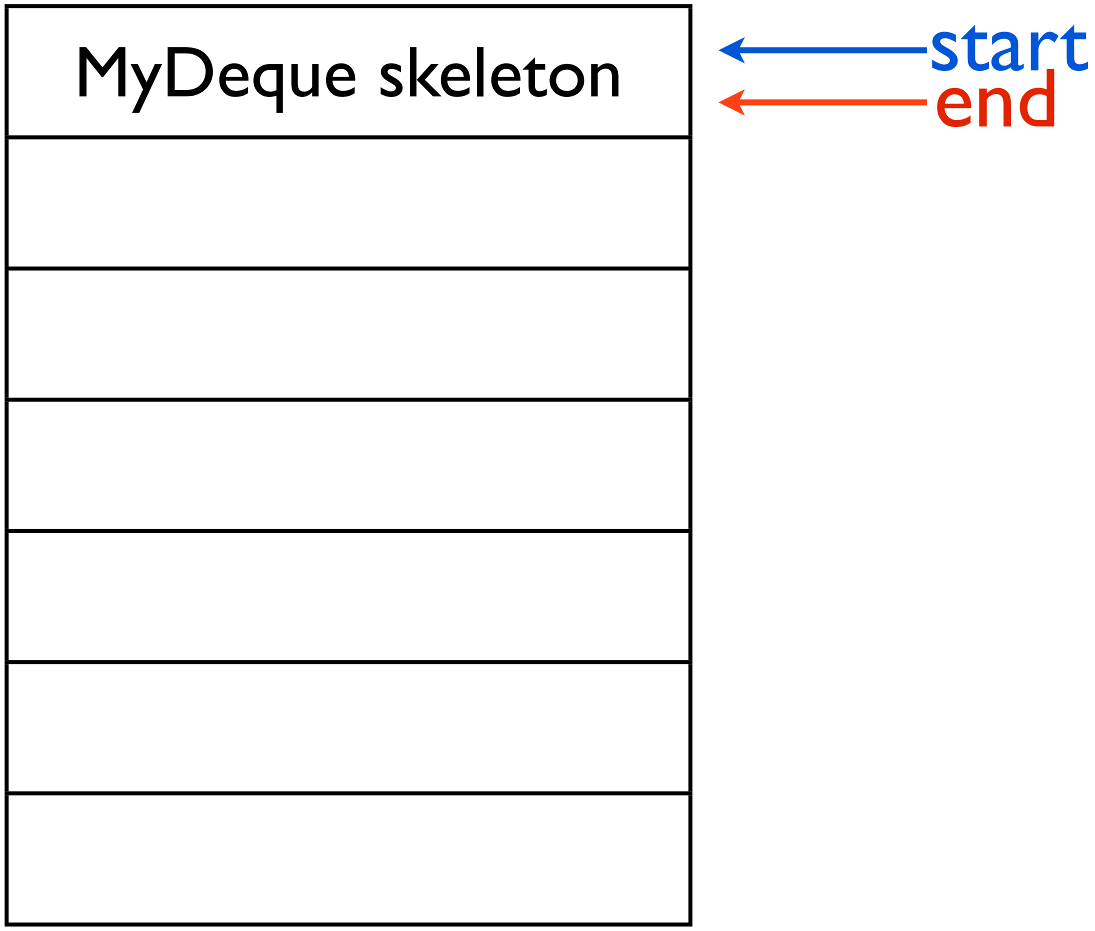
Circular Array

- Don’t shift after removing from array list
- Keep track of start and end of queue
- When run out of space, wrap around; modular arithmetic
- When array is full, increase size using list tactic

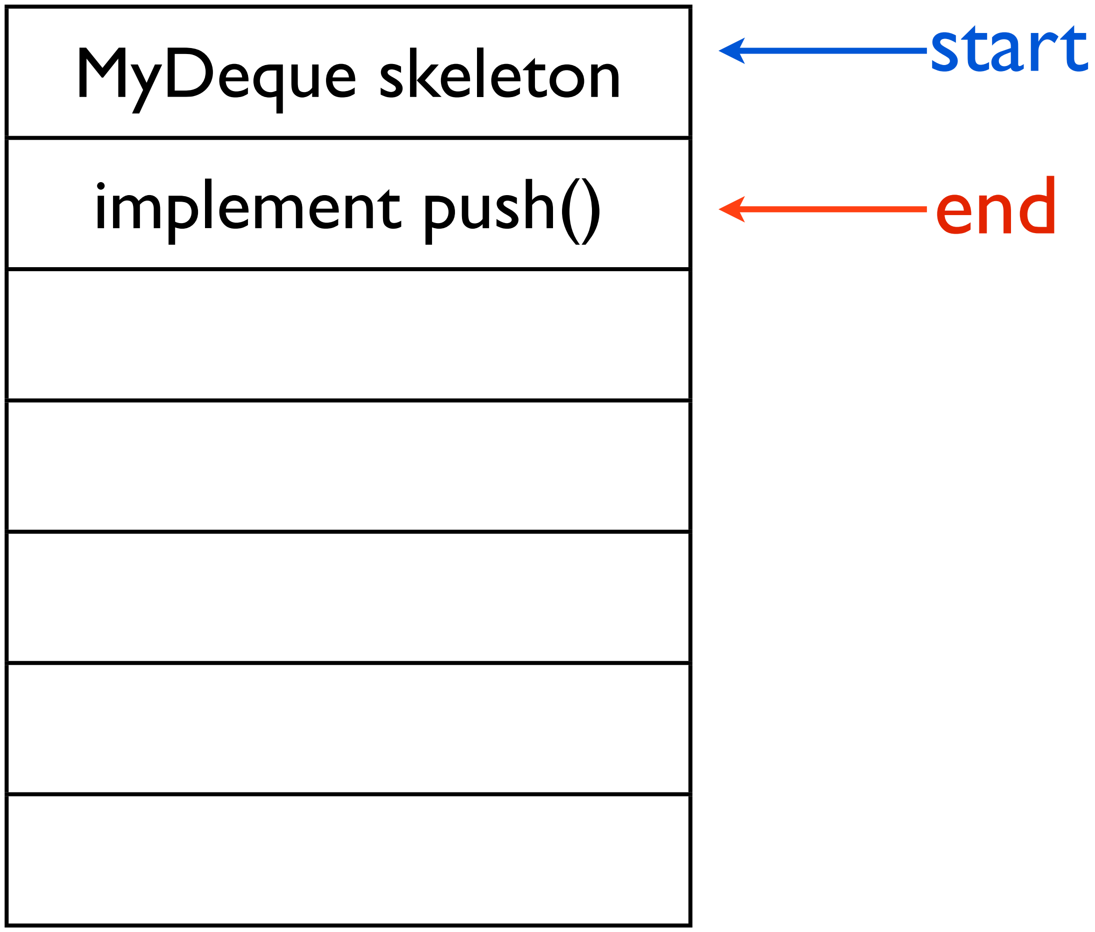
Circular Array Example



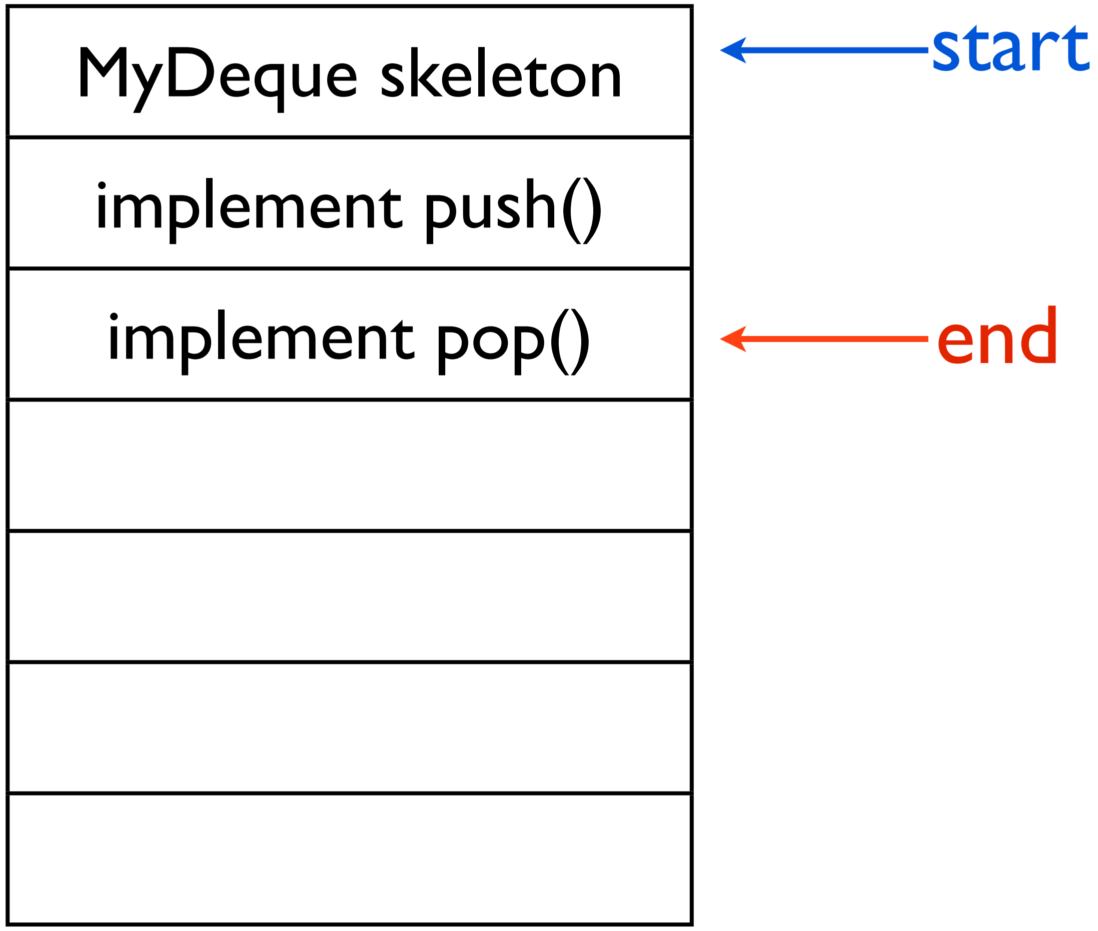
Circular Array Example



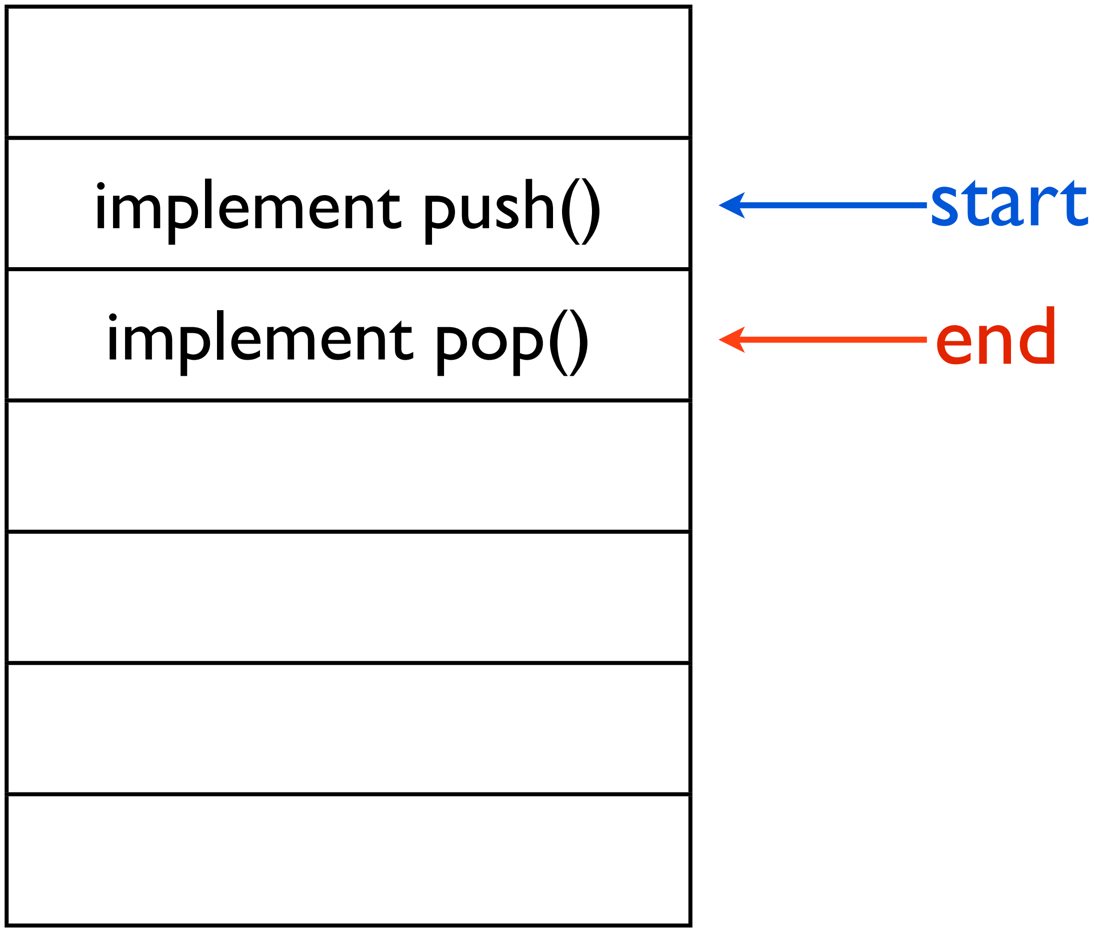
Circular Array Example



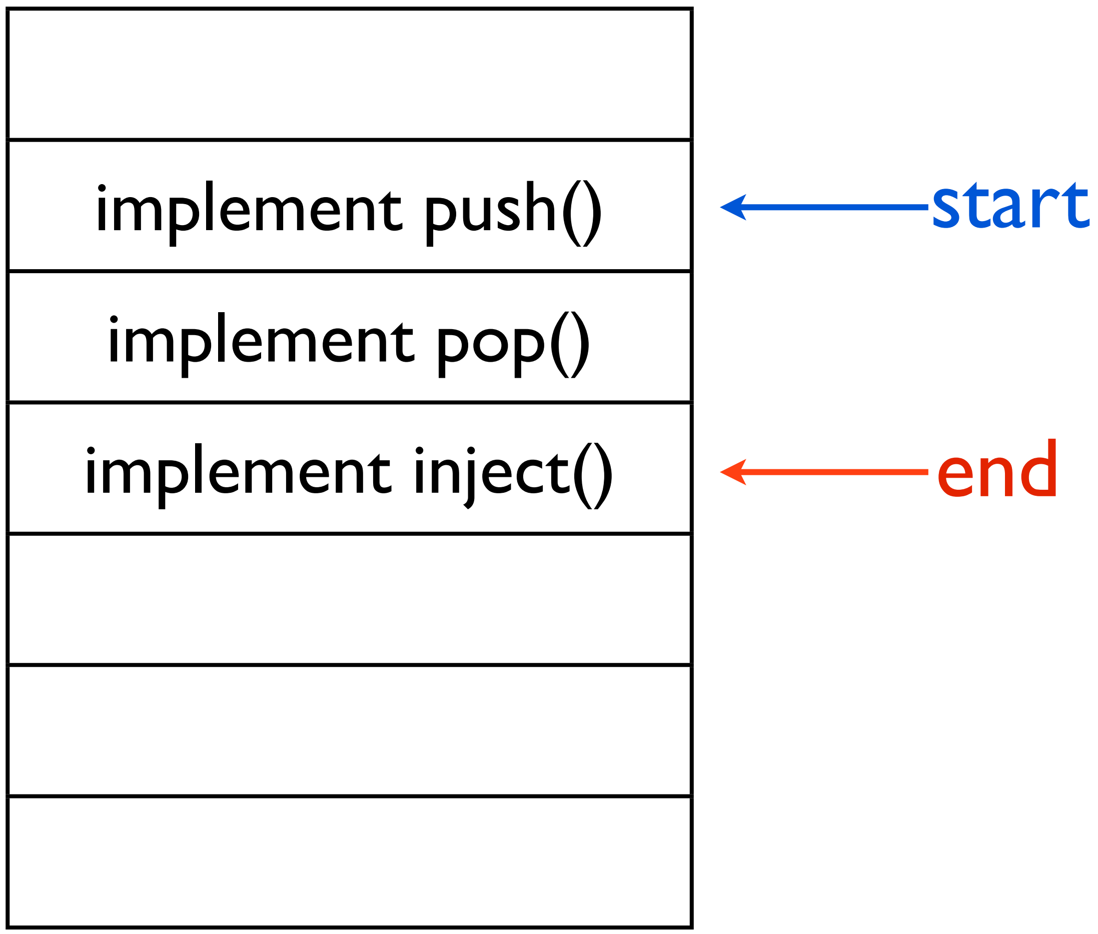
Circular Array Example



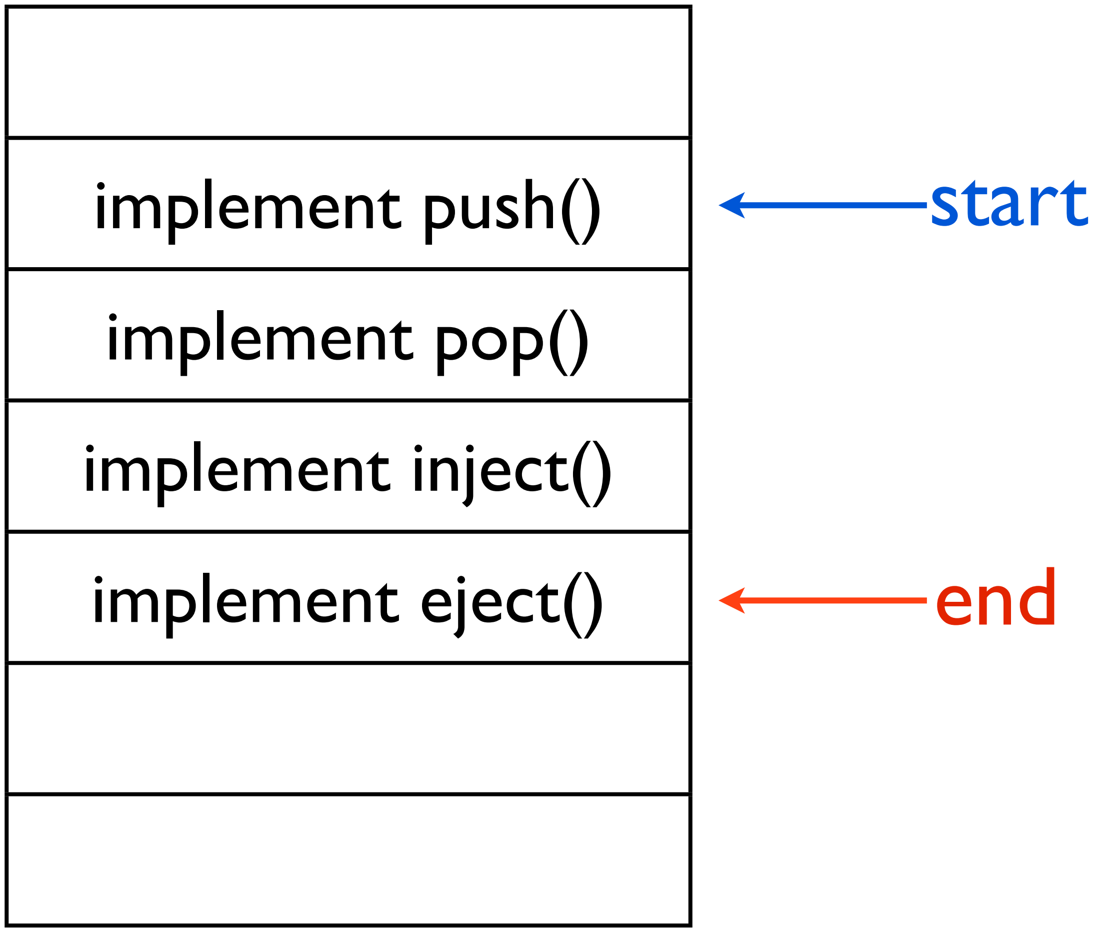
Circular Array Example



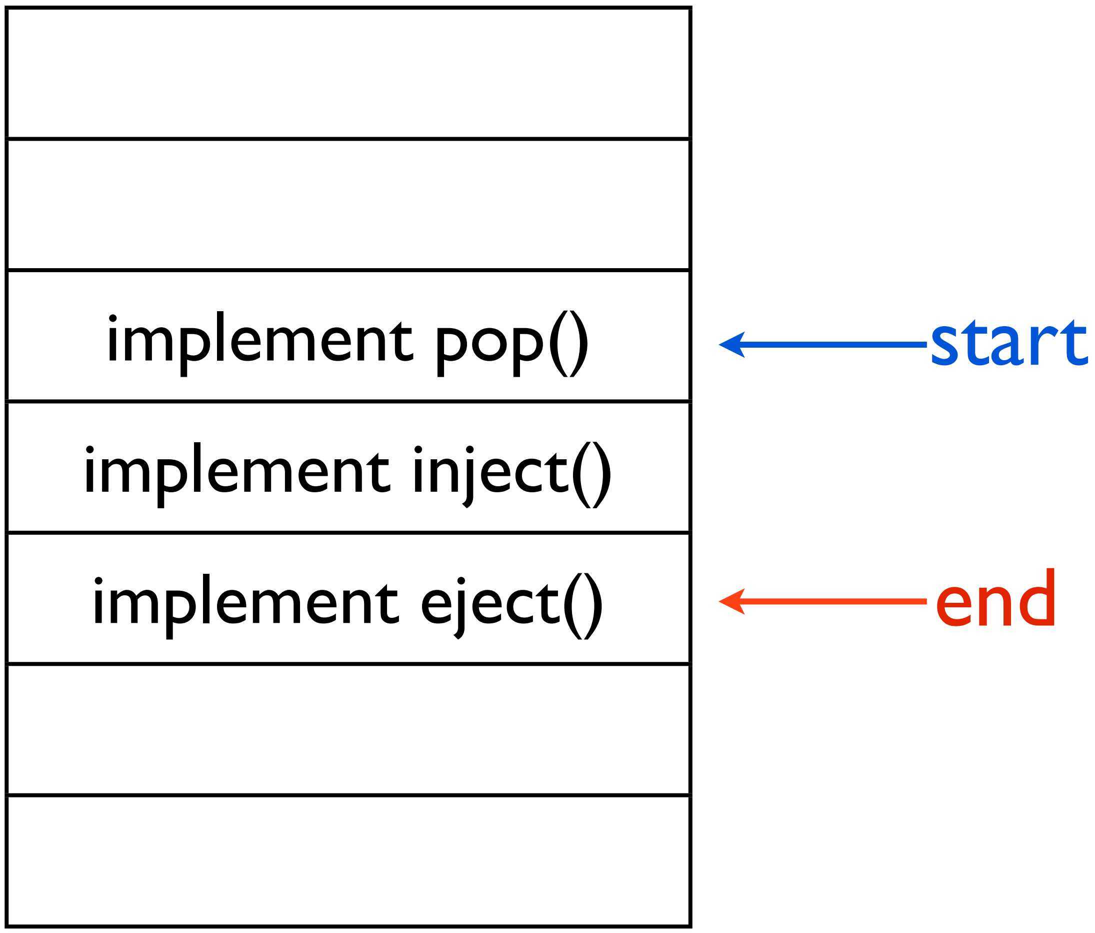
Circular Array Example



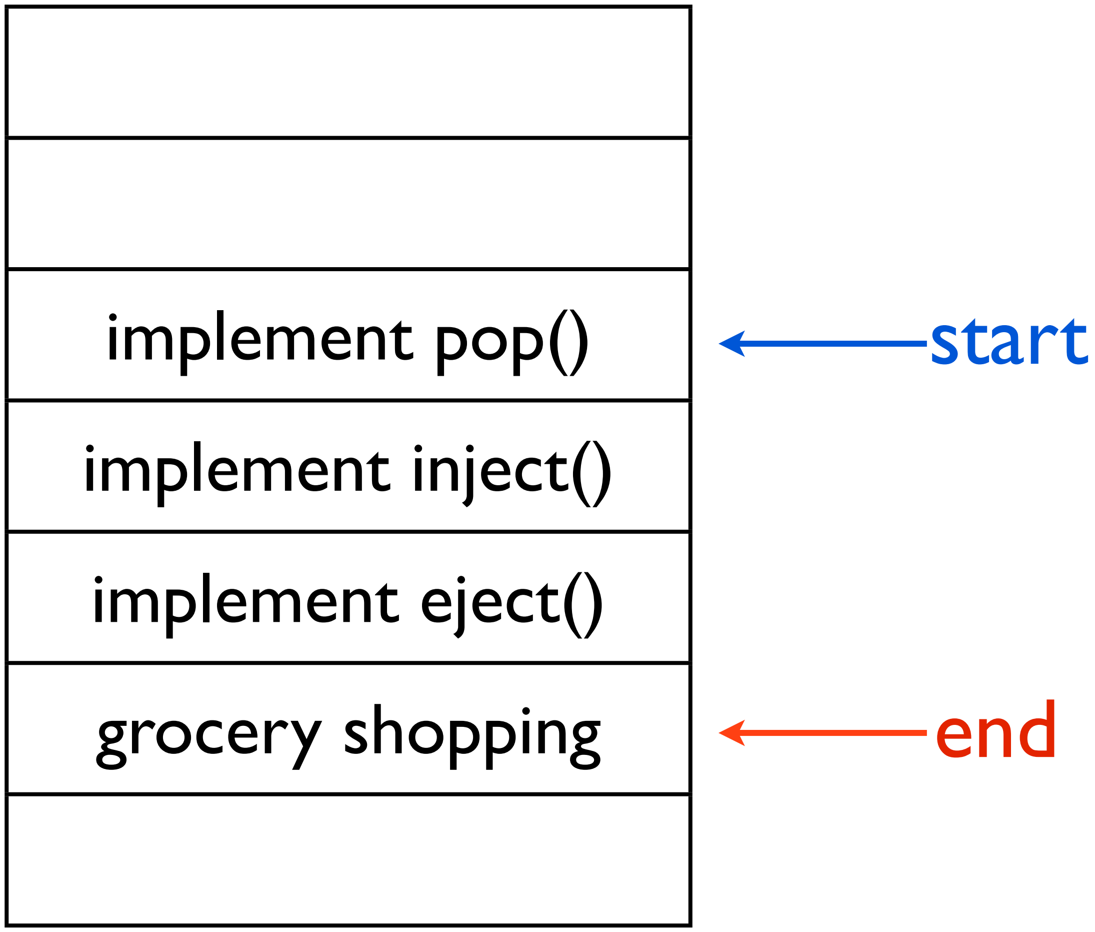
Circular Array Example



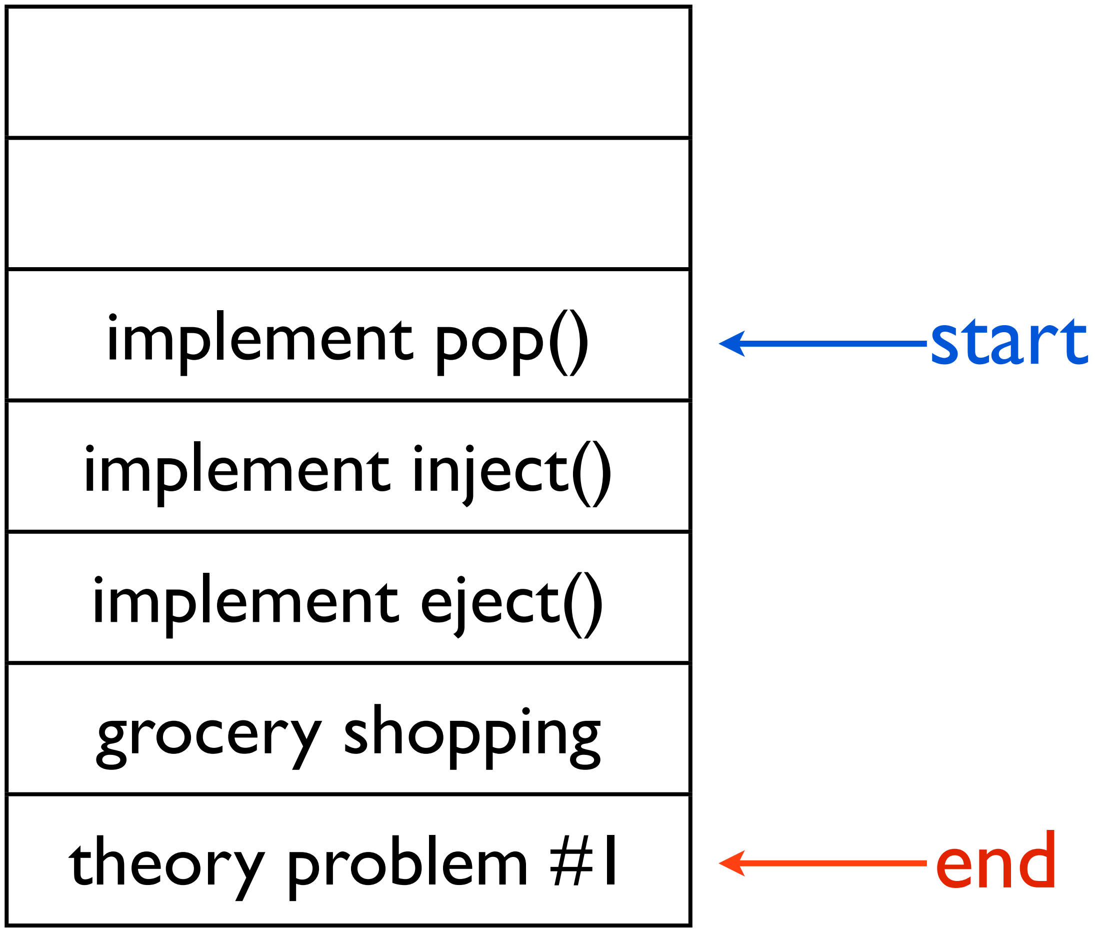
Circular Array Example



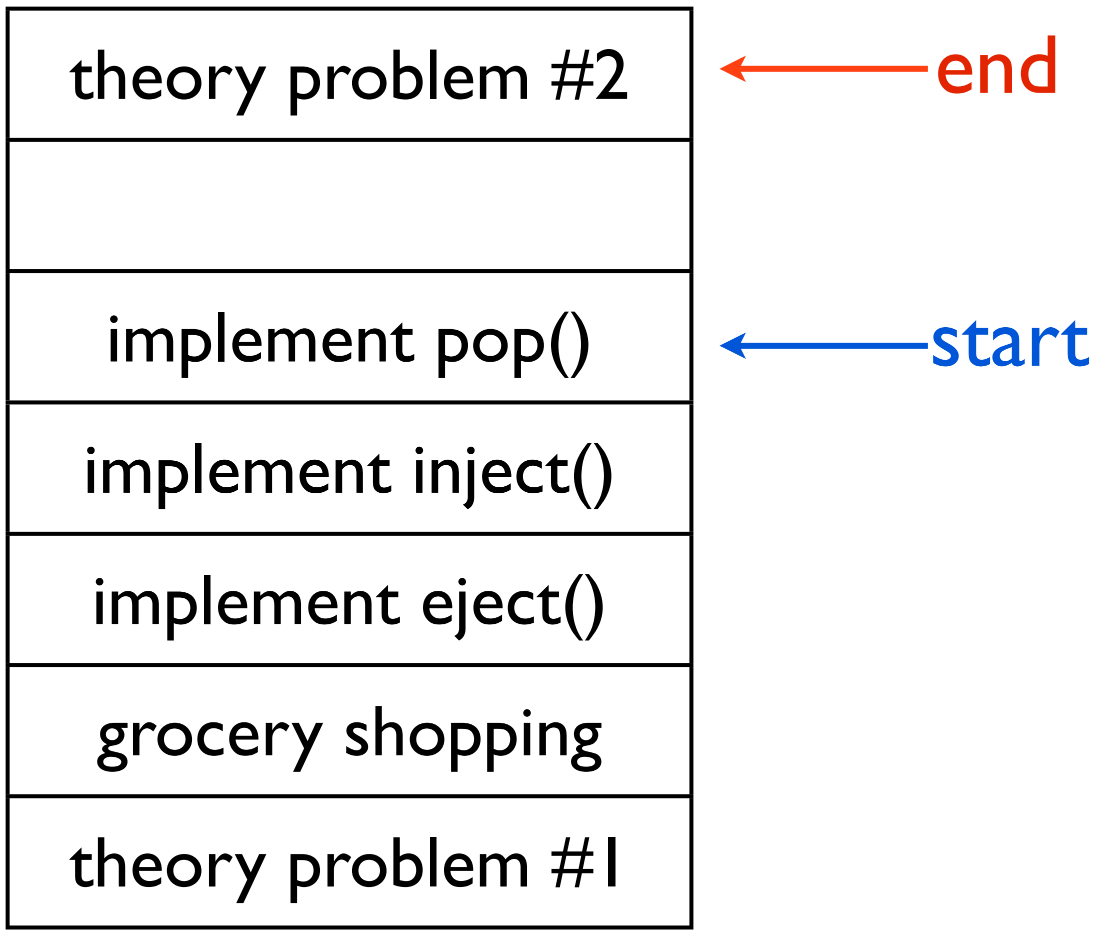
Circular Array Example



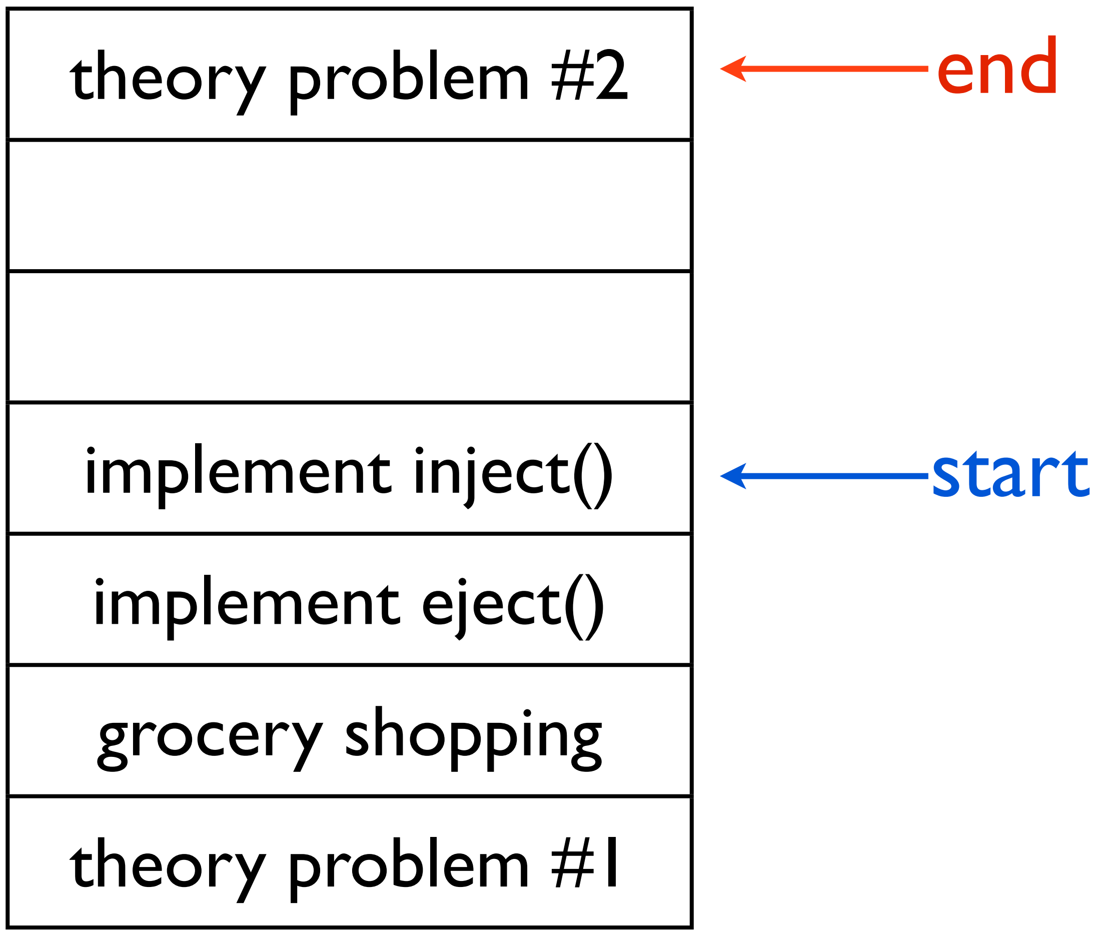
Circular Array Example



Circular Array Example



Circular Array Example



Reading

- Stacks and Queues: Weiss 3.6-3.7