Stacks

adt revisited
- we're only concerned with the interface (how to use the data structure)
- we're not concerned with the implementation details
- we may not even have access to all of the stored data items (there may be restrictions)
- a stack is an adt

- arrays are data storage structures
- so are linked lists
- a stack is a specialized structure that we as programmers use as a tool to help us solve problems

definition
- a container that allows push and pop operations
- push: add an item to the top of the container
- pop: remove the item from the top of the container (returns its value)
- so insertion and deletion occurs at one end
- think about a sink full of "stacked" dishes
- so we only have access to the last item inserted
- this is a LIFO (last-in-first-out) data structure (or FILO)

uses
- when running your programs
  - do you ever get a stack overflow exception?
- compilers utilize stacks to compile code

typical operations
- Push()
- Pop()
- Peek()
- Size()
- IsEmpty()
- IsFull()

several examples
- reversing the word: TOILET (other interesting words: LIVE, RACECAR)
  - push T
  - push O
  - push I
  - ...
  - push T

  - pop T
  - pop E
  - pop L
  - ...
  - pop T
- matching brackets and the like
  - a+b*(c+(d-e)/(f/g))

we push left parentheses
we ignore operators and operands
when we get a right parenthesis, we pop and make the match

input	stack
a	
+	
b	
*	
(	(
c	(
+	(
(	((
d	((
-	((
e	((
)	(
/	(
(	((
f	((
/	((
g	((
)	(
)	

what about (a+(b-c)?

(	(
a	(
+	(
(	((
b	((
-	((
c	((
)	(
	(error!

we could extend this to other bracket types: {}[]
we could even check for misplaced matches (e.g. [{]})

complexity
- push: O(1)
- pop: O(1)

stack vs. list
- could we use our list as the basis for a stack?
- how would we implement Push()?
- what about Pop()?
- how are Size(), IsEmpty(), and IsFull() different, if at all?