Prefix notation in action

- Scheme/LISP and other functional languages tend to use a prefix notation

```scheme
(define (square x) (* x x))

(define (expt b n)
  (if (= n 0)
    1
    (* b (expt b (- n 1)))))
```
Postfix notation in action

- Practical example of use of stack abstraction
- Put operator after operands in expression
  - Use stack to evaluate
    - operand: push onto stack
    - operator: pop operands push result
- PostScript is a stack language mostly used for printing
  - drawing an “X” with two equivalent sets of code

%! 200 200 moveto
100 100 rlineto
200 300 moveto
100 -100 rlineto
stroke showpage

%! 100 -100 200 300 100 100 200 200
moveto rlineto moveto rlineto
stroke showpage
Postfix notation in action

- For arithmetic expressions, is there more than one Postfix representation?

\[ A + B - C / D * E \]

\[ A * B * C / D * E \]
Parentheses Matching Problem

- How can we use a stack to check the syntactic correctness of expressions involving parentheses?
  
  ```java
  if (msg.equals(txt.substring(3, n - 2 + txt.length())))
  ```

- Is there a simple solution not using stacks?

- What about including braces, brackets, angle-brackets, etc.?
  
  ```java
  x = m[k] + w[msg.indexOf(s[n])];
  ```
Queue: another linear ADT

- **FIFO**: first in, first out, used in many applications
  - Scheduling jobs/processes on a computer
  - Tenting policy?
  - Computer simulations

- **Common operations**
  - Add to back, remove from front, peek at front
    - No standard java.util.Queue, instead java.util.LinkedList
    - addLast(), getFirst(), removeFirst, size()
    - Can use add() rather than addLast();

- **Downside of using LinkedList as queue**
  - Can access middle elements, remove last, etc. why?
Stack and Queue implementations

- Different implementations of queue (and stack) aren’t really interesting from an algorithmic standpoint
  - Complexity is the same, performance may change (why?)
  - Use ArrayList, growable array, Vector, linked list, …
    - Any sequential structure

- As we'll see java.util.LinkedList is good basis for all
  - In Java 5, LinkedList implements the new Queue interface

- ArrayList for queue is tricky, ring buffer implementation, add but wrap-around if possible before growing
  - Tricky to get right (exercise left to reader)
Using linear data structures

- We’ve studied arrays, stacks, queues, which to use?
  - *It depends* on the application
  - ArrayList is multipurpose, why not always use it?
    - Make it clear to programmer what’s being done
    - Other reasons?

- Other linear ADTs exist
  - List: add-to-front, add-to-back, insert anywhere, iterate
    - Alternative: create, head, tail, Lisp or
    - Linked-list nodes are concrete implementation
  - Deque: add-to-front, add-to-back, random access
    - Why is this “better” than an ArrayList?
    - How to implement?