

EECS 12: Lecture 9

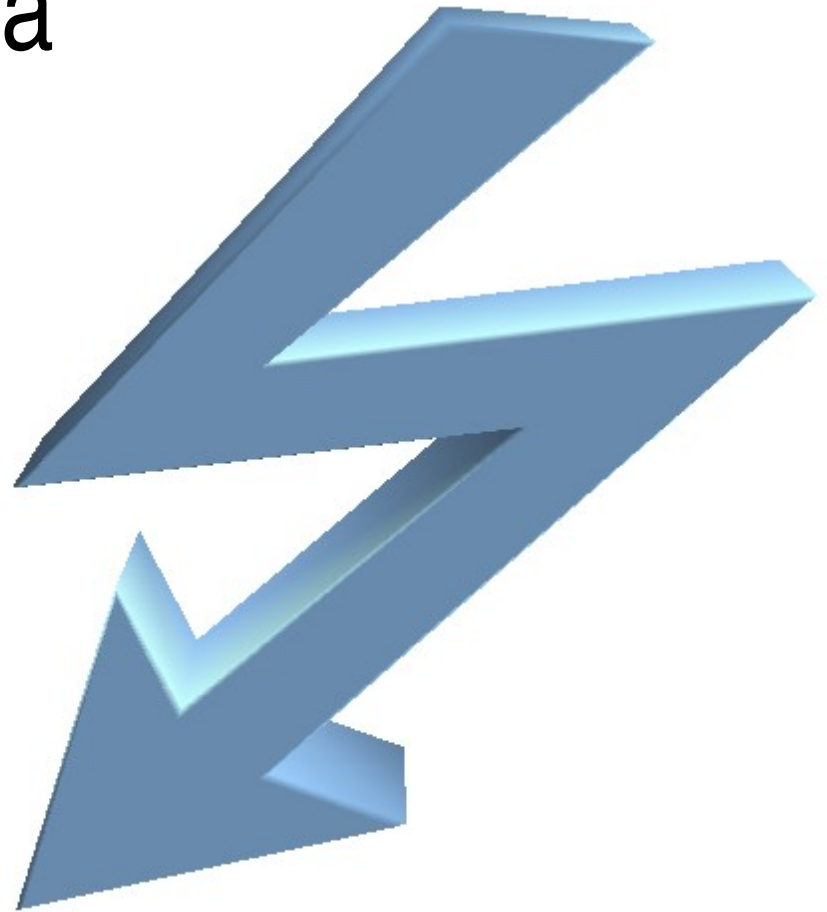
Linked Lists and Stacks

Mark E. Phair
mphair@gmail.com
UC Irvine EECS

July 31st, 2006

Agenda

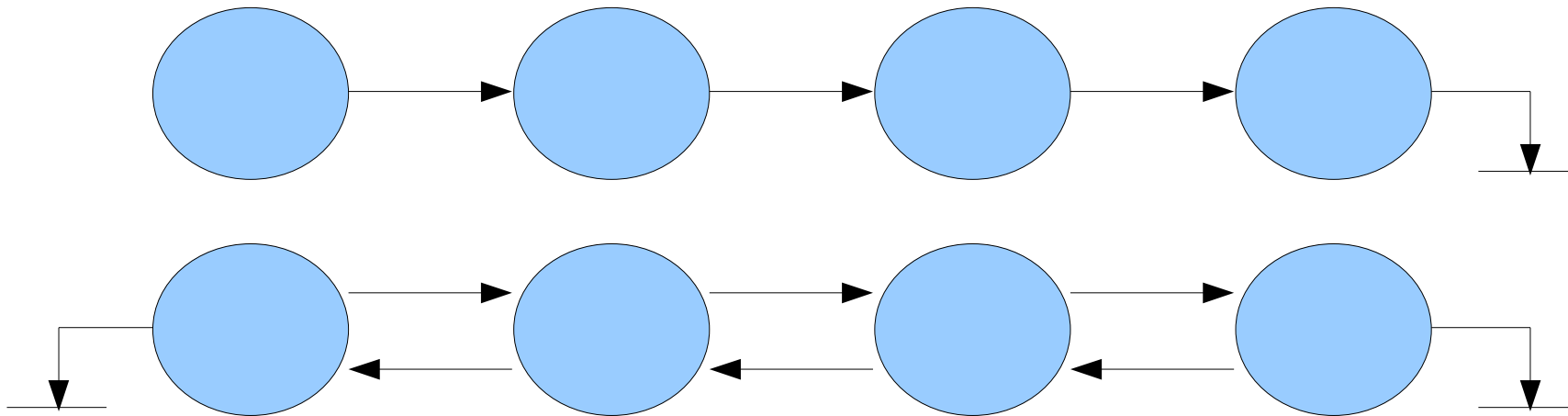
- Linked Lists
- `break` and `continue`
- Stacks versus queues
- Postfix calculators
- Very basic regex
- Python factoid of the day



She turned me into a stack!
A stack?
I got better...

Linked Lists are boring trees

- Trees have data and multiple children
- Linked lists have data and a *next*
- *Doubly linked lists* also have a *previous*



Singly Linked List Node Implementation

```
class LLNode:
    def __init__(self, data, next):
        self.data = data
        self.next = next
    def __str__(self):
        return str(self.data)
```

Doubly Linked List Node Implementation

```
class LLNode:
    def __init__(self, data, prev, next):
        self.data = data
        self.prev = prev
        self.next = next
    def __str__(self):
        return str(self.data)
```

Find a Node in a LL using *tail recursion*

(doesn't work currently in python and Guido said that it won't...)

```
def find(item, lst):  
    if lst.data == item:  
        return lst  
    elif lst.next == None:  
        return None  
    else:  
        return find(item, lst.next)
```

Find a Node in a LL using iteration

```
def find(item, lst):  
    node = lst  
    while node != None:  
        if node.data == item:  
            return node  
        node = node.next  
    return None
```

Insert After

```
def insertAfter(toInsert, item, lst):  
    node = find(item, lst)  
    if node == None:  
        return False  
    else:  
        toInsert.next = node.next  
        node.next = toInsert  
        return True
```

Let's think...

- Can we use `insertAfter` to insert at the beginning of a list?
- How would we have to change `insertAfter` if we were working with a doubly-linked list instead?

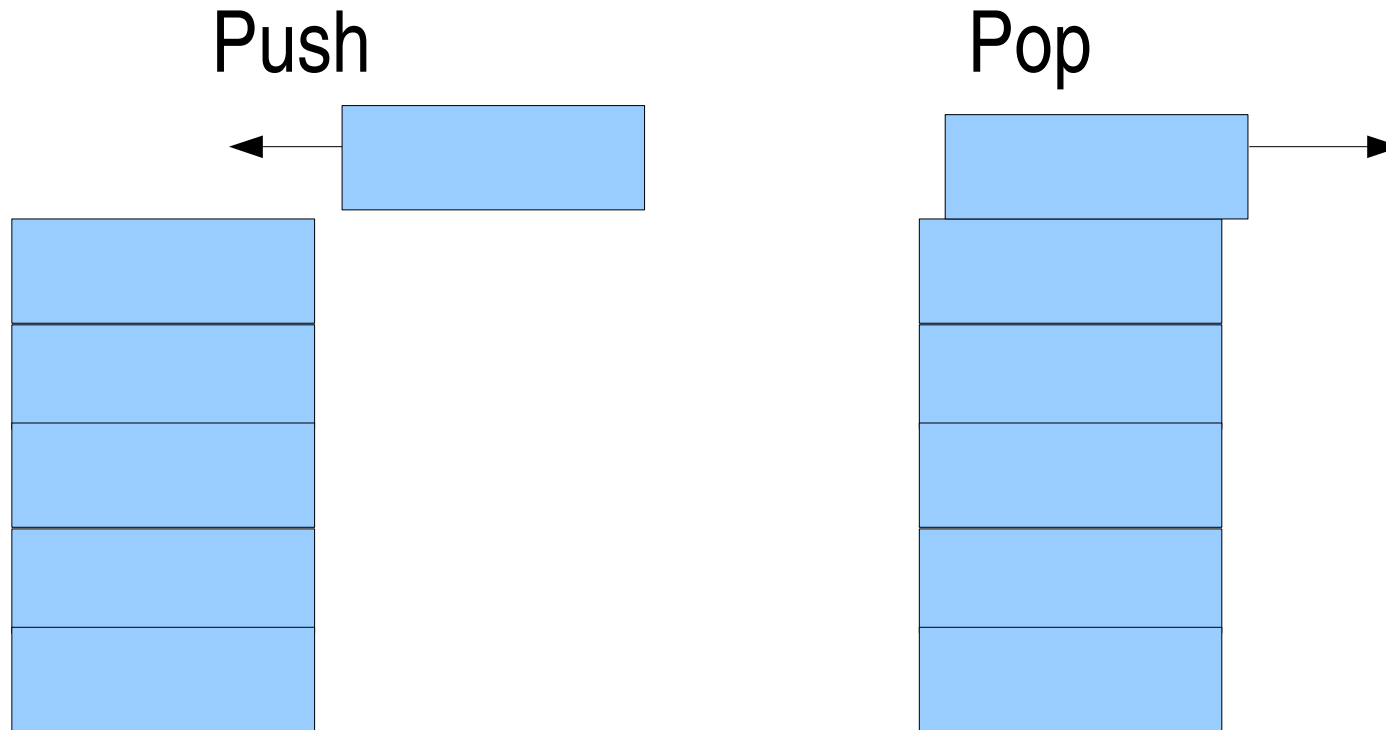
Aside: break and continue

```
>>> for ltr in ['a', 'b', 'c', 'd', 'e']:
    if ltr == 'b':
        continue # go on to next
    elif ltr == 'd':
        break # stop looping
    print ltr,
```

a c

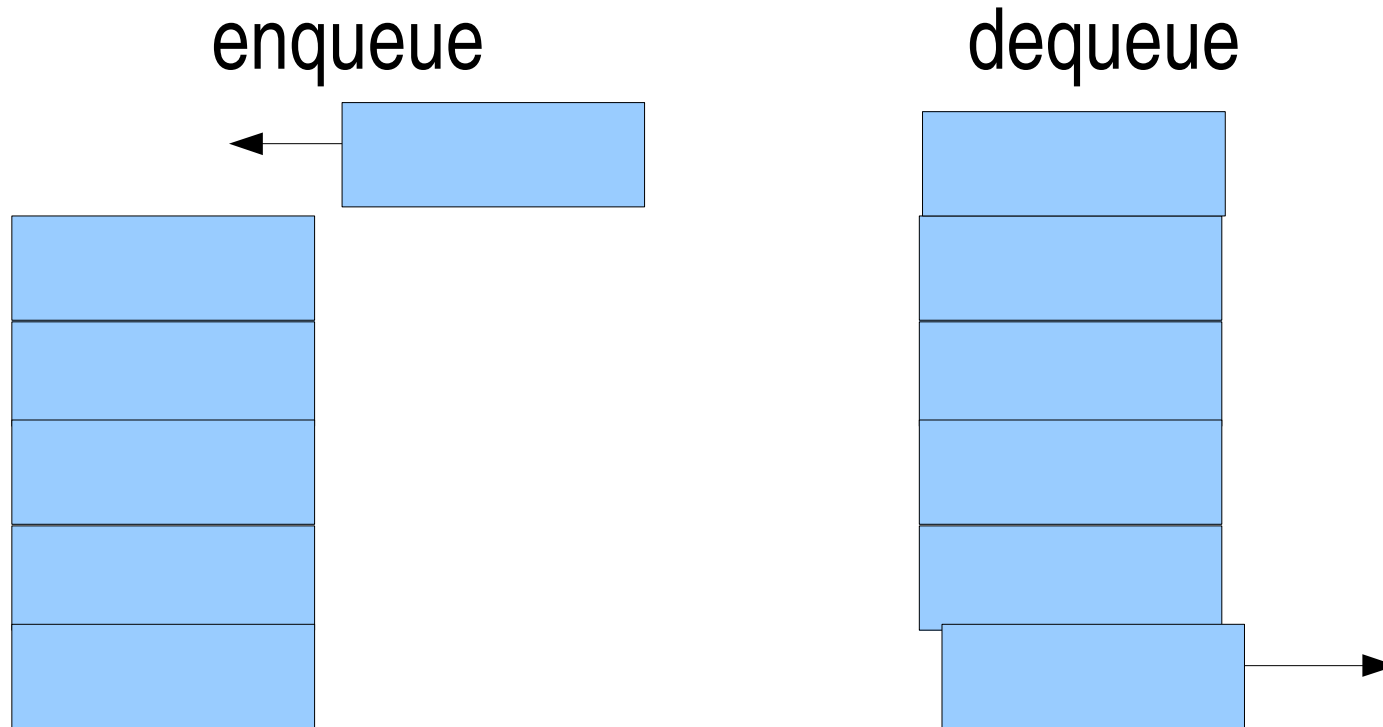
Stacks

First In Last Out (FILO) or Last In First Out (LIFO)



Queues

First In First Out (FIFO)



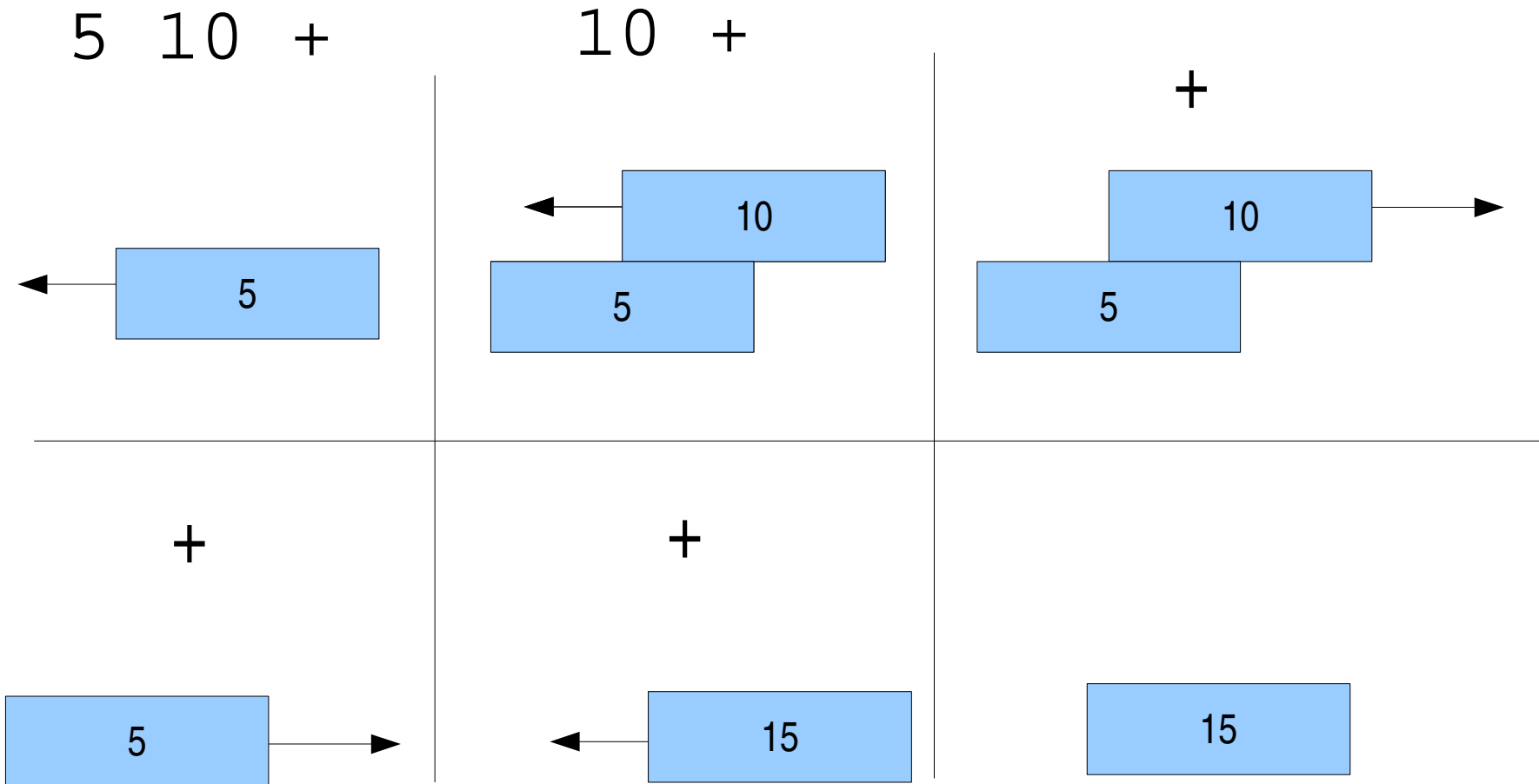
Stack: list-based implementation

```
class Stack(list):  
    def push(self, item):  
        self.append(item)  
  
    # pop is already part of list  
  
    def __str__(self):  
        return 'Stack( '\n  
                + str(list(self)) \n  
                + ' ) '
```

Postfix calculators

- Called “Reverse Polish Notation”
 - Prefix notation was developed by Polish mathematician Jan Lukasiewicz
 - Charles L. Hamblin devised postfix notation
- Operators come after operands
- Easy to parse with a stack

Postfix add



Let's work it out...

Compute the value of the following postfix expressions

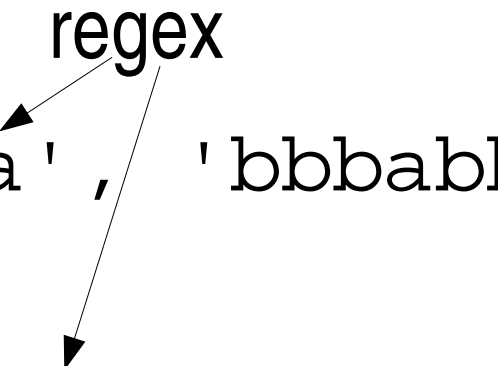
(hint: postfix $a\ b\ -$: infix $a\ -\ b$)

- $2\ 4\ +\ 7\ *$
- $47\ 5\ -\ 47\ +$
- $27\ 3\ /\ 13\ +\ 3\ *$

Simple Regular Expressions

A regular expression (regex) is a *pattern* that can be matched in a string. We can use it to split strings

```
>>> import re      regex
>>> re.split('a', 'bbbabbb')
['bbb', 'bbb']
>>> re.split('(a)', 'bbbabbb')
['bbb', 'a', 'bbb']
```



Simple Regular Expressions (continued)

The “one of these” construct []

```
>>> re.split('[ac]', 'bbbabcbb')
```

```
['bbb', 'b', 'bb']
```

```
>>> re.split('([ac])', 'bbbabcbb')
```

```
['bbb', 'a', 'b', 'c', 'bb']
```

Simple Regular Expressions (continued)

The “none of these” construct [^]

```
>>> re.split("[^ac]", 'babcb')
```

```
['', 'a', 'c', '']
```

```
>>> re.split('([^ac])', 'babcb')
```

```
['', 'b', 'a', 'b', 'c', 'b', '']
```

Simple Regular Expressions (continued)

All letters and all numbers [A-z] [0-9]

```
>>> re.split('[A-z]', 'a1b2c3')  
['', '1', '2', '3']
```

```
>>> re.split('[0-9]', 'a1b2c3')  
['a', 'b', 'c', '']
```

Simple Regular Expressions (continued)

`findall` matches instead of `splits`

```
>>> re.findall( '[A-z]', 'a1b2c3' )
```

```
['a', 'b', 'c']
```

```
>>> re.findall( '[0-9]', 'a1b2c3' )
```

```
['1', '2', '3']
```

Let's explore...

- Develop a single regex that matches both letters and numbers
- Develop a single regex that matches anything that is neither a letter nor a number

Python Factoid of the day: `import this` and the Zen of Python

```
>>> import this
```

```
[...]
```

```
>>> this.s
```

```
[...]
```

```
>>> this.d
```