CS240
Fall 2014

Mike Lam, Professor

The Little Boat

The storm tossed the little boat like a cheap sneaker in an old washing machine. The three drunken fishermen were used to such treatment, of course, but not the tree salesman, who even as a stowaway now felt that he had overpaid for the voyage.

1. Will the salesman die?
2. What color is the boat?
3. And what about Naomi?

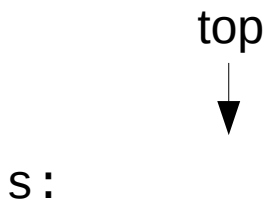
Stacks

Stacks

- Last in, first out (LIFO) sequence data structu
- Basic operations
  - `S.push(e)` add element e to top
  - `S.pop()` remove and return top element
  - `S.top()` return (but do not remove) top element
  - `S.is_empty()` return True if no elements
  - `len(S)` return number of elements

Stacks

- `s = Stack()`

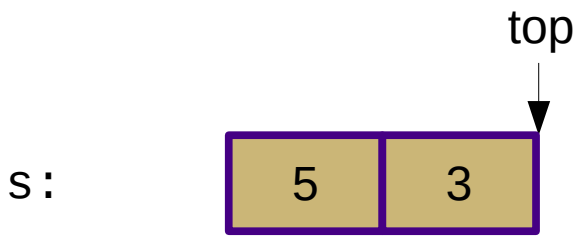


Stacks

- `s.push(5)`

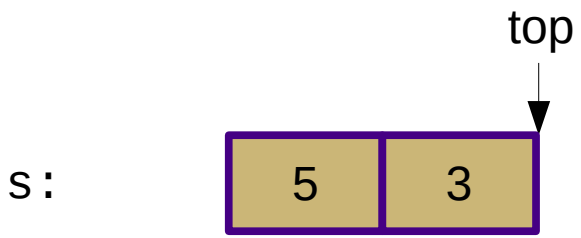
Stacks

- `s.push(3)`



Stacks

- `len(s) == 2`



Stacks

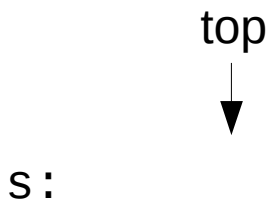
- `s.pop() == 3`

Stacks

- `s.is_empty() == False`

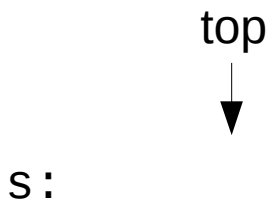
Stacks

- `s.pop() == 5`



Stacks

- `s.is_empty() == True`



Stack Implementation

- Using Array from PA2
  - `from t_array import Array`
  - Creation: `a = Array(<capacity>)`
  - Get length:`len(a)`
  - Access:`a[i]`
  - Modify:`a[i] = x`
  - Clean up:`a.free()`

Stack Analysis

Operation	Running Time
`S.push(x)`	
`S.pop()`	
`S.top()`	
`S.is_empty()`	
`len(S)`	

* = amortized

Stack Analysis

Operation	Running Time
`S.push(x)`	$O(1)$*
`S.pop()`	$O(1)$*
`S.top()`	$O(1)$
`S.is_empty()`	$O(1)$
`len(S)`	$O(1)$

* = amortized

Stacks

- Applications
  - Reversing a list
  - Storing browser history
  - Storing undo actions
  - Recursion (calling stack)
    - Grows "downward" in memory!

Application: Bracket Matching

- Problem: Check for matching parentheses "()", brackets "[]", and braces "{}"
- Hard to do with simple iteration
  - How to keep track of what we're trying to match?
- Use a stack!

Application: Bracket Matching

- Problem: Check for matching parentheses "()", brackets "[]", and braces "{}"
- Algorithm:
  - for each letter in text
    - if letter in LEFT_OPS
      - stack.push(letter)
    - if letter in RIGHT_OPS
      - if stack.is_empty() or letter != stack.pop()
        - return False
  - return stack.is_empty()

Application: Postfix Notation

- Postfix notation
  - Also referred to as "Reverse Polish Notation" (RP
- Normal "infix" notation: 2 + 3
- Postfix notation: 2 3 +
- Prefix notation: + 2 3
- Why is postfix notation interesting?

Application: Postfix Notation

- Infix notation is hard to evaluate
- Consider: `2 + 3 * 4`
  - `(2 + 3) * 4 = 20`
  - `2 + (3 * 4) = 14`
- Need to evaluate the "*" first
  - How to tell this without looking ahead?
  - Use postfix notation: `3 4 * 2 +`

Application: Postfix Notation

- Evaluate: `3 4 * 2 +`
- Algorithm:
  - for each word in expression
    - if word is a number
      - stack.push(word)
    - if word is an operator
      - op1 = stack.pop()
      - op2 = stack.pop()
      - stack.push(op1 <op> op2)
  - return stack.pop()