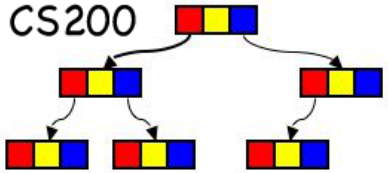
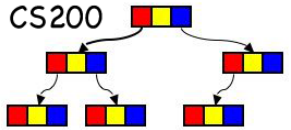


CS200: Stacks

- Prichard Ch. 7

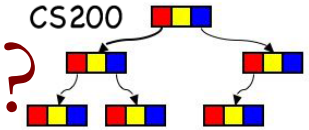
Linear, time-ordered structures



- Data structures that reflect a *temporal* relationship
  - order of removal based on order of insertion
- We will consider:
  - "first come,first serve"
    - first in first out - FIFO (queue)
  - "take from the top of the pile"
    - last in first out - LIFO (stack)

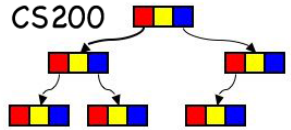
Stacks or queues?

What can we do with coin dispenser?

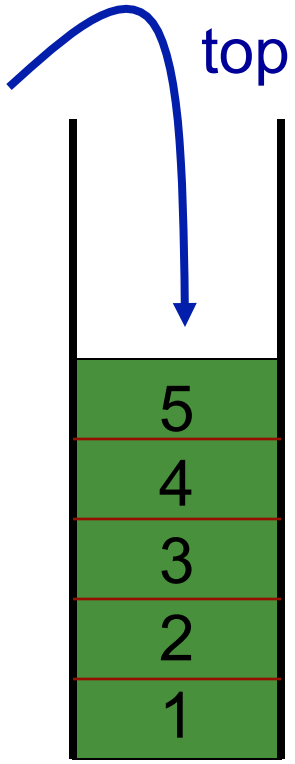


- "**push**" a coin into the dispenser.
- "**pop**" a coin from the dispenser.
- "**peek**" at the coin on top, but don't pop it.
- "**isEmpty**" check whether this dispenser is empty or not.

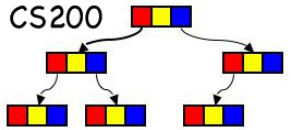
Stacks



- Last In First Out (LIFO) structure
  - A stack of dishes in a cafe
- Add/Remove done from same end: the top

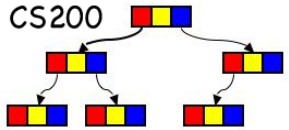


Possible Stack Operations



- **`isEmpty()`**: determine whether stack is empty
- **`push()`**: add a new item to the stack
- **`pop()`**: remove the item added most recently
- **`peek()`**: retrieve the item added most recently

Checking for balanced braces

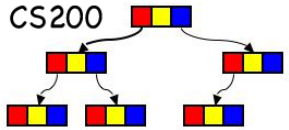


- How can we use a stack to determine whether the braces in a string are balanced?

```
abc{defg{ijk}{l{mn}}op}qr
```

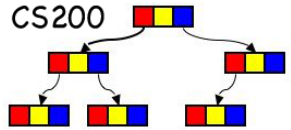
```
abc{def}}{ghij{kl}m
```

Pseudocode



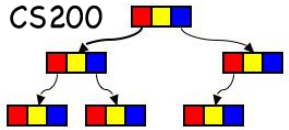

```
while ( not at the end of the string){
      if (the next character is a “{“){
            aStack.push(“{“)
      }
      else if (the character is a “}”) {
            if(aStack.isEmpty()) ERROR!!!
            else aStack.pop()
      }
}
if(!aStack.isEmpty()) ERROR!!!
```

Expressions



- Types of Algebraic Expressions
  - Prefix
  - Postfix (RPN)
  - Infix
- Prefix and postfix are easier to parse. No ambiguity.
- Postfix: operator applies to the operands that immediately precede it.
- Examples:
  1. 5 4 3 * -
  2. 5 * 4 - 3
  3. * - 5 4 3

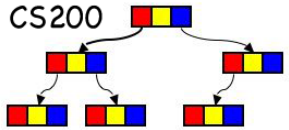
operands are written in the conventional way



What type of expression is “5 * 4 3 –”?

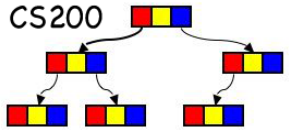
A. Prefix
B. Infix
C. Postfix
D. None of the above (i.e., illegal)

Evaluating a Postfix Expression

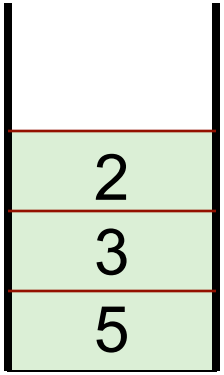



```
while there are input tokens left
    read the next token
    if the token is a value
        push it onto the stack.
    else
        //the token is a operator taking n arguments
        pop the top n values from the stack and perform the operation
        push the result on the stack
If there is only one value in the stack return it as the result
else
    throw an exception
```

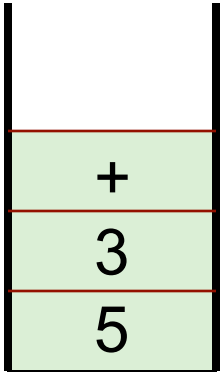
Quick check



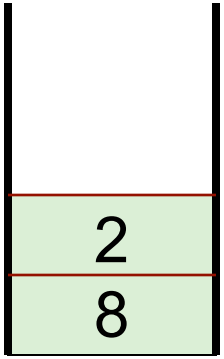
- If the input string is “5 3 + 2 *”, which of the following could be what the stack looks like when trying to parse it?



A

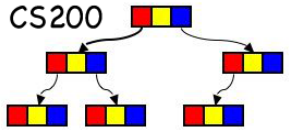


B



C

Stack Interface



push(StackItemType newItem)

- adds a new item to the top of the stack

StackItemType pop() throws StackException

- deletes the item at the top of the stack and returns it
- Exception when deletion fails

StackItemType peek() throws StackException

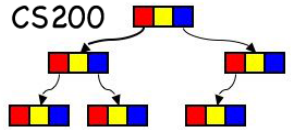
- returns the top item from the stack, but does not remove it
- Exception when retrieval fails

boolean isEmpty()

- returns true if stack empty, false otherwise

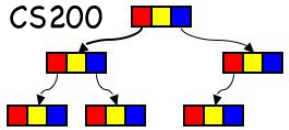
Preconditions? Postconditions?

Comparison of Implementations



- Options for Implementation:
  - Array based implementation
  - ArrayList based implementation
  - Reference based implementation
- What are the advantages and disadvantages of each implementation?
- Let's look at an Linked List based implementation
- In P1 you implement an ArrayList based implementation

Stack API in Java

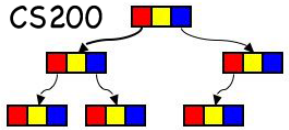


`public class` **Stack<E>** `extends` Vector<E>

Implemented Interfaces: Iterable<E>, Collection<E>, List<E>, RandomAccess

- `Stack` extends `Vector` with operations that allow a vector to be treated as a stack (`push, pop, peek, empty, search`)

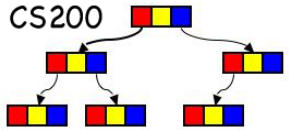
Stacks and Recursion



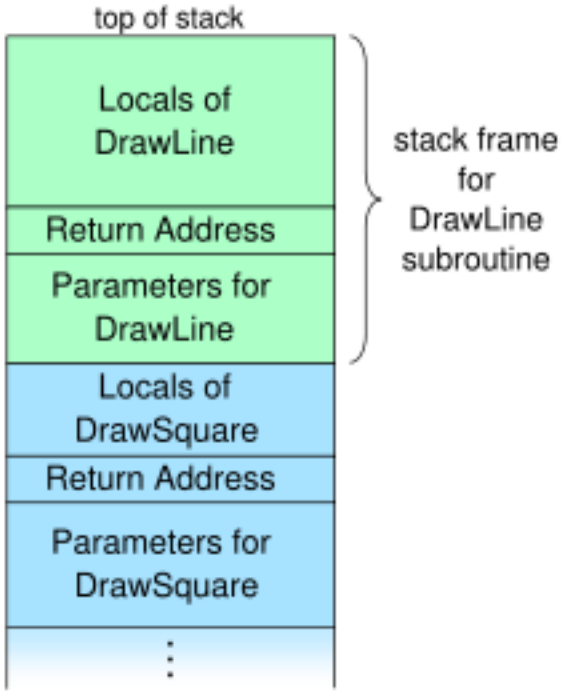
- Most implementations of recursion maintain a stack of activation records.

- Within recursive calls, the most recently executed activation record is stored at the top of the stack.

Applications - the run-time stack



- Nested method calls tracked on call stack (aka run-time stack)
  - First method that returns is the last one invoked
- Element of call stack - activation record
  - parameters
  - local variables
  - return address: pointer to next instruction to be executed in calling method



http://en.wikipedia.org/wiki/Image:Call_stack_layout.svg