Stacks
The Stack ADT

The Stack ADT stores arbitrary objects

Insertions and deletions follow the last-in first-out scheme

Think of a spring-loaded plate dispenser

Main stack operations:

- push(object): inserts an element
- T pop(): removes and returns the last inserted element
- T peek(): returns the last inserted element without removing it
- boolean empty(): indicates whether no elements are stored
Exceptions

- Attempting the execution of an operation of ADT may sometimes cause an error condition, called an exception.
- Exceptions are said to be “thrown” by an operation that cannot be executed.

In the Stack ADT, operations pop and peek cannot be performed if the stack is empty.

We’d like these methods to throw an EmptyStackException.
### Stack Example

**Notation:** (TOS, ..., BOS)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Returns</th>
<th>$S$</th>
</tr>
</thead>
<tbody>
<tr>
<td>push(5)</td>
<td>–</td>
<td>(5)</td>
</tr>
<tr>
<td>push(3)</td>
<td>–</td>
<td>(3, 5)</td>
</tr>
<tr>
<td>pop()</td>
<td>3</td>
<td>(5)</td>
</tr>
<tr>
<td>push(7)</td>
<td>–</td>
<td>(7, 5)</td>
</tr>
<tr>
<td>pop()</td>
<td>7</td>
<td>(5)</td>
</tr>
<tr>
<td>top()</td>
<td>5</td>
<td>(5)</td>
</tr>
<tr>
<td>pop()</td>
<td>5</td>
<td>()</td>
</tr>
<tr>
<td>pop()</td>
<td>“error”</td>
<td>()</td>
</tr>
<tr>
<td>empty()</td>
<td>true</td>
<td>()</td>
</tr>
<tr>
<td>push(9)</td>
<td>–</td>
<td>(9)</td>
</tr>
<tr>
<td>push(7)</td>
<td>–</td>
<td>(7, 9)</td>
</tr>
<tr>
<td>push(3)</td>
<td>–</td>
<td>(3, 7, 9)</td>
</tr>
<tr>
<td>push(5)</td>
<td>–</td>
<td>(5, 3, 7, 9)</td>
</tr>
<tr>
<td>pop()</td>
<td>5</td>
<td>(3, 7, 9)</td>
</tr>
</tbody>
</table>
Example Applications of Stacks

- Postfix calculator
- Infix calculator
- Page-visited history in a Web browser
- Parenthesis matching
- HTML parsing
Postfix calculator

- Each input is either a number or an arithmetic operator
- An operator applies to the top 2 items on the stack
- Example: \(1 \ 2 \ + \ 3 \ 4 \ * \ * = 36\)

Algorithm:
- Number -> push
- Operator -> pop, pop, calculate, push
Web history

- Two stacks (for back and forward buttons)
- Each new visited page is pushed on the "back" stack
- Pushing back or forward buttons: pop off one stack, push onto the other
Parentheses Matching

Each “(”, “{”, or “[” must be paired with a matching “)”, “}”, or “[”

- correct: ( )( ) )( ( ( ) ) )
- correct: ( ( ( ) ( ) ) ) ( ( ( ) ) )
- incorrect: ) ( ( ) ( ( ) )
- incorrect: [ [ ] ]
- incorrect: ( ( ) ) ( ) ( ) ( )
Parentheses Matching Algorithm

Algorithm ParenMatch($X, n$):

**Input:** An array $X$ of $n$ tokens, each of which is either a grouping symbol, a variable, an arithmetic operator, or a number

**Output:** true if and only if all the grouping symbols in $X$ match

Let $S$ be an empty stack

for $i = 0$ to $n - 1$ do

  if $X[i]$ is an opening grouping symbol then
  
  * $S$.push($X[i]$)

  else if $X[i]$ is a closing grouping symbol then

  * if $S$.isEmpty() then
    
    return false {nothing to match with}

  * if $S$.pop() does not match the type of $X[i]$ then
    
    return false {wrong type}

  if $S$.isEmpty() then

  * return true {every symbol matched}

  else

  * return false {some symbols were never matched}
For fully-correct HTML, each <name> should pair with a matching </name>

<body>
  <center>
    <h1> The Little Boat </h1>
  </center>
  <p> 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. </p>
  <ol>
    <li> Will the salesman die? </li>
    <li> What color is the boat? </li>
    <li> And what about Naomi? </li>
  </ol>
</body>

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?