Stacks

- Stack ADT
- Stack Implementations
- Stack Applications
Stacks

- A restricted access container that stores a linear collection.
  - Very common for solving problems in computer science.
  - Provides a last-in first-out (LIFO) protocol.
The Stack

- New items are added and existing items are removed from the top of the stack.
The Stack ADT

- A stack stores a linear collection of items with access limited to a last-in first-out order.
  - Adding and removing items is restricted to the top of the stack.

- \textbf{Stack()}
- isEmpty()
- length()
- pop()
- peek()
- push( item )
Stack Example

```
# Extracts a collection of integer values from the user
# and prints them in reverse order.

PROMPT = "Enter an int value (<0 to end):"
myStack = Stack()

# Extract the values and push them onto a stack.
value = int(input( PROMPT ))
while value >= 0 :
    myStack.push( value )
    value = int(input( PROMPT ))

# Pop the values from the stack and print each.
while not myStack.isEmpty() :
    value = myStack.pop()
    print( value )
```
Stack Implementation

- Several common ways to implement a stack:
  - Python list
    - easiest to implement
  - Linked list
    - better choice when a large number of push and pop operations are performed.
Stack: Python List

- How is the data organized within the Python list?
  - Most efficient is to let the end of the list represent the top of the stack.

*Why would the reverse organization not be as efficient?*
# Implementation of the Stack ADT using a Python list.
class Stack:
    def __init__(self):
        self._theItems = list()

    def isEmpty(self):
        return len(self) == 0

    def __len__(self):
        return len(self._theItems)

    def peek(self):
        assert not self.isEmpty(), "Cannot peek at an empty stack"
        return self._theItems[-1]

    def pop(self):
        assert not self.isEmpty(), "Cannot pop from an empty stack"
        return self._theItems.pop()

    def push(self, item):
        self._theItems.append(item)
Stack: Linked List

- How should the data be organized?
  - Let the head of the list represent the top of the stack.

Why is this the most efficient organization?
class Stack:
    def __init__(self):
        self._top = None
        self._size = 0

    def isEmpty(self):
        return self._top is None

    def __len__(self):
        return self._size

    def peek(self):
        assert not self.isEmpty(), "Cannot peek at an empty stack"
        return self._top.item

# ...  # The private storage class for creating stack nodes.
class _StackNode:
    def __init__(self, item, link):
        self.item = item
        self.next = link
class Stack:
    # ...

    def pop(self):
        assert not self.isEmpty(), "Cannot pop from an empty stack"
        node = self._top
        self.top = self._top.next
        self._size -= 1
        return node.item

    def push(self, item):
        self._top = _StackNode(item, self._top)
        self._size += 1
Stack Applications

- Many applications encountered in computer science requires the use of a stack.
  - Balanced delimiters
  - Postfix expressions
Balanced Delimiters

- Many applications use delimiters to group strings of text or simple data into subparts.
  - mathematical expressions
  - programming languages
  - HTML markup
Consider the following C++ source code:

```cpp
int sumList( int theList[], int size )
{
    int sum = 0;
    int i = 0;
    while( i < size ) {
        sum += theList[ i ];
        i += 1;
    }
    return sum;
}
```
Source Code Example

- The delimiters must be paired and balanced.
- We can design and implement an algorithm to:
  - scan a C++ source file, and
  - determine if the delimiters are properly paired.
Valid C++ Source?

```python
from lliststack import Stack

def isValidSource( srcfile ):
    s = Stack()
    for line in srcfile :
        for token in line :
            if token in "{{[" :
                s.push( token )
            elif token in "]})" :
                if s.isEmpty() :
                    return False
                left = s.pop()
                if (token == ")" and left != ")") or 
                   (token == "]" and left != "]") or 
                   (token == ")" and left != ")") :
                    return False
    return s.isEmpty()
```
Mathematical Expressions

- We work with mathematical expressions on a regular basis.
  - Easy to determine the order of evaluation.
  - Easy to calculate.
- But the task is more difficult in computer programs.
  - A program can not visualize the expression to determine the order of evaluation.
  - Must examine one token at a time.
Types of Expressions

- Three different notations can be used:
  - infix: $A + B \times C$
  - prefix: $+ A \times B C$
  - postfix: $A B C \times +$
Infix to Postfix

- Infix expressions can be easily converted by hand to postfix notation.

\[ A \times B + C / D \]

1. Fully parenthesize the expression.

\[ ((A \times B) + (C / D)) \]

2. For each set of (), move operator to the end of the closing parenthesis.

\[ ((A B \times) (C D /) +) \]
Infix to Postfix (cont)

- The expression at the end of step 2:
  
  \[( (A \ B \ *) \ (C \ D \ /) \ +)\]

  3. Remove all of the parentheses.

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

- Which results in the postfix version.
Evaluating Postfix Expressions

- We can evaluate a valid postfix expression using a stack structure.
- For each token:
  - If the current token is an operand, push its value onto the stack.
  - If the current token is an operator:
    - pop the top two operands off the stack.
    - perform the operation (top value is RHS operand).
    - push the result of the operation back on the stack.
- The final result will be the last value on the stack.
Postfix Evaluation Examples

- To illustrate the use of the algorithm, assume
  - the existence of an empty stack, and
  - the following variable assignments

\[
\begin{align*}
A &= 8 \\
B &= 2 \\
C &= 3 \\
D &= 4
\end{align*}
\]

- Evaluate the valid expression:
  \[
  A \ B \ C \ + \ * \ D \ / 
  \]
# Postfix Example #1

<table>
<thead>
<tr>
<th>Token</th>
<th>Alg Step</th>
<th>Stack</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC+*D/</td>
<td>1</td>
<td>8</td>
<td>push value of A</td>
</tr>
<tr>
<td>ABC+*D/</td>
<td>1</td>
<td>8 2</td>
<td>push value of B</td>
</tr>
<tr>
<td>ABC+*D/</td>
<td>1</td>
<td>8 2 3</td>
<td>push value of C</td>
</tr>
<tr>
<td>ABC+*D/</td>
<td>2(a)</td>
<td>8</td>
<td>pop top two values: y = 3, x = 2</td>
</tr>
<tr>
<td></td>
<td>2(b)</td>
<td>8</td>
<td>compute z = x + y or z = 2 + 3</td>
</tr>
<tr>
<td></td>
<td>2(c)</td>
<td>8 5</td>
<td>push result (5) of the addition</td>
</tr>
<tr>
<td>ABC+*D/</td>
<td>2(a)</td>
<td></td>
<td>pop top two values: y = 5, x = 8</td>
</tr>
<tr>
<td></td>
<td>2(b)</td>
<td></td>
<td>compute z = x * y or z = 8 * 5</td>
</tr>
<tr>
<td></td>
<td>2(c)</td>
<td>40</td>
<td>push result (40) of the multiplication</td>
</tr>
<tr>
<td>ABC+*D/</td>
<td>1</td>
<td>40 4</td>
<td>push value of D</td>
</tr>
<tr>
<td>ABC+*D/</td>
<td>2(a)</td>
<td></td>
<td>pop top two values: y = 4, x = 40</td>
</tr>
<tr>
<td></td>
<td>2(b)</td>
<td></td>
<td>compute z = x / y or z = 40 / 4</td>
</tr>
<tr>
<td></td>
<td>2(c)</td>
<td>10</td>
<td>push result (10) of the division</td>
</tr>
</tbody>
</table>
Postfix Example #2

- What happens if the expression is invalid?

\[ \text{A B} \ast \text{C D} + \]

<table>
<thead>
<tr>
<th>Token</th>
<th>Alg Step</th>
<th>Stack</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB*CD+</td>
<td>1</td>
<td>8</td>
<td>push value of A</td>
</tr>
<tr>
<td>AB*CD+</td>
<td>1</td>
<td>8 2</td>
<td>push value of B</td>
</tr>
<tr>
<td>AB*CD+</td>
<td>2(a)</td>
<td></td>
<td>pop top two values: y = 2, x = 8</td>
</tr>
<tr>
<td></td>
<td>2(b)</td>
<td></td>
<td>compute ( z = x \ast y ) or ( z = 8 \ast 2 )</td>
</tr>
<tr>
<td></td>
<td>2(c)</td>
<td>16</td>
<td>push result (16) of the multiplication</td>
</tr>
<tr>
<td>AB*CD+</td>
<td>1</td>
<td>16 3</td>
<td>push value of C</td>
</tr>
<tr>
<td>AB*CD+</td>
<td>1</td>
<td>16 3 4</td>
<td>push value of D</td>
</tr>
<tr>
<td>AB*CD+</td>
<td>2(a)</td>
<td>16</td>
<td>pop top two values: y = 4, x = 3</td>
</tr>
<tr>
<td></td>
<td>2(b)</td>
<td>16</td>
<td>compute ( z = x + y ) or ( z = 3 + 4 )</td>
</tr>
<tr>
<td></td>
<td>2(c)</td>
<td>16 7</td>
<td>push result (7) of the addition</td>
</tr>
<tr>
<td>Error</td>
<td>xxxxxx</td>
<td>xxxxx</td>
<td>Too many values left on the stack.</td>
</tr>
</tbody>
</table>
### Postfix Example #3

- What happens if there are too many operators for the given number of operands? \[ A \ B \ * \ + \ C \ / \]

<table>
<thead>
<tr>
<th>Token</th>
<th>Alg Step</th>
<th>Stack</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB*+C/</td>
<td>1</td>
<td>8</td>
<td>push value of A</td>
</tr>
<tr>
<td>AB*+C/</td>
<td>1</td>
<td>8 2</td>
<td>push value of B</td>
</tr>
<tr>
<td>AB*+C/</td>
<td>2(a)</td>
<td></td>
<td>pop top two values: ( y = 2, x = 8 )</td>
</tr>
<tr>
<td></td>
<td>2(b)</td>
<td></td>
<td>compute ( z = x \times y ) or ( z = 8 \times 2 )</td>
</tr>
<tr>
<td></td>
<td>2(c)</td>
<td>16</td>
<td>push result (16) of the multiplication</td>
</tr>
<tr>
<td>AB*+C/</td>
<td>2(a)</td>
<td></td>
<td>pop top two values: ( y = 16, x = ? )</td>
</tr>
<tr>
<td>Error</td>
<td>xxxxxx</td>
<td>xxxxx</td>
<td>Only one value on stack, two needed.</td>
</tr>
</tbody>
</table>