Topic 15
Implementing and Using Stacks

"stack n.
The set of things a person has to do in the future. "I haven't
done it yet because every time I pop my stack something new
gets pushed." If you are interrupted several times in the
middle of a conversation, "My stack overflowed" means "I
forget what we were talking about."

-The Hacker's Dictionary

Friedrich L. Bauer
German computer scientist
who proposed "stack method
of expression evaluation"
in 1955.

Stacks

- Access is allowed only at one point of the structure,
normally termed the top of the stack
  - access to the most recently added item only
- Operations are limited:
  - push (add item to stack)
  - pop (remove top item from stack)
  - top (get top item without removing it)
  - isEmpty
- Described as a "Last In First Out" (LIFO) data structure

Stack Operations

Assume a simple stack for integers.
Stack<Integer> s = new Stack<Integer>();
s.push(12);
s.push(4);
s.push( s.top() + 2 );
s.pop();
s.push( s.top() );
//what are contents of stack?
Stack Operations

Write a method to print out contents of stack in reverse order.

Uses of Stacks

- The runtime stack used by a process (running program) to keep track of methods in progress
- Search problems
- Undo, redo, back, forward

What is Output?

Stack<Integer> s = new Stack<Integer>();
// put stuff in stack
for(int i = 0; i < 5; i++)
    s.push(i);
// Print out contents of stack
// while emptying it.
// Assume there is a size method.
for(int i = 0; i < s.size(); i++)
    System.out.print(s.pop() + " ");

A 0 1 2 3 4   D 2 3 4
B 4 3 2 1 0   E No output due
c 4 3 2       to runtime error

Corrected Version

Stack<Integer> s = new Stack<Integer>();
// put stuff in stack
for(int i = 0; i < 5; i++)
    s.push(i);
// print out contents of stack
// while emptying it
int limit = s.size();
for(int i = 0; i < limit; i++)
    System.out.print(s.pop() + " ");
// or
// while( !s.isEmpty() )
//     System.out.println(s.pop());
Implementing a stack

- need an underlying collection to hold the elements of the stack
- 2 obvious choices
  - array (native or ArrayList)
  - linked list
- Adding a *layer of abstraction*. A big idea.
- array implementation
- linked list implementation

Applications of Stacks

Mathematical Calculations

- What does \(3 + 2 \times 4\) equal?
  - \(2 \times 4 + 3\)?
  - \(3 \times 2 + 4\)?
- The precedence of operators affects the order of operations.
- A mathematical expression cannot simply be evaluated left to right.
- A challenge when evaluating a program.
- *Lexical analysis* is the process of interpreting a program.

What about \(1 - 2 - 4 \div 5 \times 3 \times 6 \div 7 \div 2 \div 3\)

Infix and Postfix Expressions

- The way we are use to writing expressions is known as *infix notation*
- Postfix expression does not require any precedence rules
- \(3 \times 2 \times 1 +\) is postfix of \(3 \times 2 + 1\)
- evaluate the following postfix expressions and write out a corresponding infix expression:
  - \(2 \times 3 \times 2 \times 4 + +\)
  - \(1234^* +\)
  - \(12 - 32^* 3 \div 6 +\)
  - \(25^1-\)
Clicker Question 2

What does the following postfix expression evaluate to?

\[ 6 \ 3 \ 2 \ + \ * \]

A. 18  
B. 36  
C. 24  
D. 11  
E. 30

Evaluation of Postfix Expressions

- Easy to do with a stack
- given a proper postfix expression:
  - get the next token
  - if it is an operand push it onto the stack
  - else if it is an operator
    - pop the stack for the right hand operand
    - pop the stack for the left hand operand
    - apply the operator to the two operands
    - push the result onto the stack
  - when the expression has been exhausted the result is the top (and only element) of the stack

Infix to Postfix

- Convert the following equations from infix to postfix:
  - \[ 2 \ ^ \ 3 \ ^ \ 3 \ + \ 5 \ * \ 1 \]
  - \[ 11 \ + \ 2 \ - \ 1 \ * \ 3 \ / \ 3 \ + \ 2 \ ^ \ 2 \ / \ 3 \]

Problems:
  - Negative numbers?
  - parentheses in expression

Infix to Postfix Conversion

- Requires operator precedence parsing algorithm
  - parse v. To determine the syntactic structure of a sentence or other utterance

Operands: add to expression
Close parenthesis: pop stack symbols until an open parenthesis appears

Operators:
  - Have an on stack and off stack precedence
  - Pop all stack symbols until a symbol of lower precedence appears. Then push the operator
End of input: Pop all remaining stack symbols and add to the expression
### Simple Example

Infix Expression: \( 3 + 2 \times 4 \)

PostFix Expression:

Operator Stack:

#### Precedence Table

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Off Stack Precedence</th>
<th>On Stack Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>*</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>/</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>^</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>(</td>
<td>20</td>
<td>0</td>
</tr>
</tbody>
</table>

### Simple Example

Infix Expression: \( + 2 \times 4 \)

PostFix Expression: \( 3 \)

Operator Stack:

#### Precedence Table

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Off Stack Precedence</th>
<th>On Stack Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>*</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>/</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>^</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>(</td>
<td>20</td>
<td>0</td>
</tr>
</tbody>
</table>

### Simple Example

Infix Expression: \( 2 \times 4 \)

PostFix Expression: \( 3 \)

Operator Stack: \( + \)

#### Precedence Table

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Off Stack Precedence</th>
<th>On Stack Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>*</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>/</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>^</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>(</td>
<td>20</td>
<td>0</td>
</tr>
</tbody>
</table>

### Simple Example

Infix Expression: \( \times 4 \)

PostFix Expression: \( 3 \ 2 \)

Operator Stack: \( + \)

#### Precedence Table

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Off Stack Precedence</th>
<th>On Stack Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>*</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>/</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>^</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>(</td>
<td>20</td>
<td>0</td>
</tr>
</tbody>
</table>
Simple Example
Infix Expression: 4
PostFix Expression: 3 2
Operator Stack: + *

Precedence Table

Symbol | Off Stack Precedence | On Stack Precedence
---|---|---
+ | 1 | 1
- | 1 | 1
* | 2 | 2
/ | 2 | 2
^ | 10 | 9
( | 20 | 0

Simple Example
Infix Expression: 3 2 4 *
PostFix Expression: 3 2 4 *
Operator Stack: + *

Precedence Table

Symbol | Off Stack Precedence | On Stack Precedence
---|---|---
+ | 1 | 1
- | 1 | 1
* | 2 | 2
/ | 2 | 2
^ | 10 | 9
( | 20 | 0

Simple Example
Infix Expression: 3 2 4 *
PostFix Expression: 3 2 4 *
Operator Stack: + *

Precedence Table

Symbol | Off Stack Precedence | On Stack Precedence
---|---|---
+ | 1 | 1
- | 1 | 1
* | 2 | 2
/ | 2 | 2
^ | 10 | 9
( | 20 | 0

Simple Example
Infix Expression: 3 2 4 *
PostFix Expression: 3 2 4 *
Operator Stack: + *

Precedence Table

Symbol | Off Stack Precedence | On Stack Precedence
---|---|---
+ | 1 | 1
- | 1 | 1
* | 2 | 2
/ | 2 | 2
^ | 10 | 9
( | 20 | 0
Example

11 + 2 ^ 4 ^ 3 - ((4 + 5) * 6 ) ^ 2

Show algorithm in action on above equation

Balanced Symbol Checking

- In processing programs and working with computer languages there are many instances when symbols must be balanced { }, [ ], ( )

A stack is useful for checking symbol balance. When a closing symbol is found it must match the most recent opening symbol of the same type.

- Applicable to checking html and xml tags!

Algorithm for Balanced Symbol Checking

- Make an empty stack
- read symbols until end of file
  - if the symbol is an opening symbol push it onto the stack
  - if it is a closing symbol do the following
    - if the stack is empty report an error
    - otherwise pop the stack. If the symbol popped does not match the closing symbol report an error
- At the end of the file if the stack is not empty report an error

Algorithm in practice

- list[i] = 3 * ( 44 - method( foo( list[ 2 * (i + 1) + foo( list[i - 1] ) ) / 2 * ) - list[ method(list[0])];

- Complications
  - when is it not an error to have non matching symbols?

- Processing a file
  - *Tokenization*: the process of scanning an input stream. Each independent chunk is a token.

- Tokens may be made up of 1 or more characters