CIS 343 – Structure of Programming Languages
Winter 2013

Programming Assignment #4
Programming in Ruby

Due Date: Wednesday, April 3, 2013

Problem Specification

Write a program in Ruby to perform the following two functions: 1) convert a given infix arithmetic expression to a postfix arithmetic expression and evaluate it, and 2) evaluate a given postfix arithmetic expression. Your task is to complete the methods in the `InfixPostfix` class in the supplied `InfixPostfix.rb` file.

Assume a valid expression (infix or postfix) is entered by the user. Also, assume that an infix expression can contain only integers, binary operators (+, -, *, /, %, ^), '(', and ')'. Similarly, a postfix expression can contain only integers and binary operators. Each token (integers, binary operators, or parenthesis) in the expression is separated by at least one space.

The algorithms to use for infix to postfix conversion and postfix evaluation are on the next page. Each algorithm uses a stack in support of its operations, and in each algorithm the stack is used for a different purpose.

In Ruby, you can use an array (`Array` class) for stack operations. The `Array` class in Ruby provides the following convenient methods to support stack operations:

```
# pushes the given object to the end of the array
push(obj)

# removes the last element and returns it, or nil if empty
pop()

# returns the last element in the array, or nil if empty
last()

# returns the number of elements in the array
length()

# removes all elements the array
clear()
```

The algorithm for converting an infix to postfix is as follows:

Read infix expression from left to right.
While there are more tokens in the infix expression, do the following:

- If the current token is an operand (integer), append it to postfix expression.
- If the current token is a left parenthesis, push it onto the stack.
- If the current token is an operator:
  - Pop operators (if there are any) from the stack with "stack precedence" equal or higher than the "input precedence" of the current token, and append popped operators to the postfix expression (see below for information on stack/input operator precedence).
  - Push the current token onto the stack.
- If the current token is a right parenthesis:
  - Pop operators from the stack and append them to postfix expression until a left parenthesis is at the top of the stack.
  - Pop (and discard) the left parenthesis from the stack.

Pop the remaining operators from the stack and append them to postfix expression.

The algorithm for evaluating a postfix expression is as follows:

Read postfix expression from left to right.
While there are more tokens in the postfix expression, do the following:

- If the current token is an operand (integer), push it onto the stack.
- If the current token is an operator:
  - Pop the top element into variable y.
  - Pop the next element into variable x.
  - Perform "x operator y".
  - Push the result of the calculator onto the stack.

Pop the top value of the stack. This is the result of evaluating the postfix expression.

Testing Your Program

The supplied `InfixPostfixTest.rb` file contains unit tests that test your implementation of `InfixPostfix` class in the `InfixPostfix.rb` file.

The `InfixPostfixTest.rb` file contains six tests for `infixToPostfix()` and `evaluatePostfix()` methods in the `InfixPostfix` class.

Run the `InfixPostfixTest.rb` file to test your code.

Input and Stack Precedence of Operators

Operator	Input Precedence	Stack Precedence	Associativity
+ -	1	1	Left-to-Right
* / %	2	2	Left-to-Right
^	4	3	Right-to-Left
(	5	-1	

Instructions/Deliverables

1. Upload the completed **`InfixPostfix.rb`** file on Blackboard.
    - I will use the submission date/time on Blackboard as your official submission date/time.
    - It is your responsibility to make sure the submission on Blackboard went through successfully.
2. Because of possible version issues between Ruby 1.8 and Ruby 1.9, please clearly indicate which version of Ruby is used to implement and test your program.