Evaluate Postfix (using Stacks)

The Postfix notation is used to represent algebraic expressions. The expressions written in postfix form are evaluated faster compared to infix notation as parenthesis are not required in postfix. Here we outline the basics of evaluation of postfix expressions.

Following is rough sketch of an algorithm to evaluate postfix expressions.
1) Create a stack to store operands (or values).
2) Scan the given expression and do following for every scanned element.

    2a) If the element is a number, push it into the stack
    2b) If the element is an operator, pop operands for the operator from stack. Evaluate the operator and push the result back to the stack

3) When the expression has ended, the number in the stack is the final answer

Example:
Let the given expression be "2 3 1 * + 9 -". We scan all elements one by one.
1) Scan '2', it's a number, so push it to stack. Stack contains '2'
2) Scan '3', again a number, push it to stack, stack now contains '2 3' (from bottom to top)
3) Scan '1', again a number, push it to stack, stack now contains '2 3 1'
4) Scan '*', it's an operator, pop two operands from stack, apply the * operator on operands, we get 3*1 which results in 3. We push the result '3' to stack. Stack now becomes '2 3'.
5) Scan '+', it's an operator, pop two operands from stack, apply the + operator on operands, we get 3 + 2 which results in 5. We push the result '5' to stack. Stack now becomes '5'.
6) Scan '9', it's a number, we push it to the stack. Stack now becomes '5 9'.
7) Scan '-', it's an operator, pop two operands from stack, apply the – operator on operands, we get 5 – 9 which results in -4. We push the result '-4' to stack. Stack now becomes '-4'.
8) There are no more elements to scan, we return the top element from stack (which is the only element left in stack).