Reverse Polish Notation
Dr John Waldron

1 Problem Description

Reverse Polish notation (RPN), also known as Polish postfix notation or simply postfix notation, is a mathematical notation in which operators follow their operands, in contrast to Polish notation (PN), in which operators precede their operands. It does not need any parentheses as long as each operator has a fixed number of operands. The description Polish refers to the nationality of logician Jan Lukasiewicz, who invented Polish notation in 1924. The reverse Polish scheme was proposed in 1954 by Burks, Warren, and Wright and was independently reinvented by Friedrich L. Bauer and Edsger W. Dijkstra in the early 1960s to reduce computer memory access and utilize the stack to evaluate expressions.

Design, implement, test and document an ARM Assembly Language subroutine for evaluating Reverse Polish Notation (RPN) expressions. Your solution must demonstrate problem decomposition by using appropriate subroutines.

Unlike the more familiar infix notation, where an operator appears between its two operands, Reverse Polish Notation (RPN), or postfix notation, places an operator after its operands. RPN expressions are evaluated from left to right. One advantage of RPN is that, unlike infix notation, parentheses are not required to specify precedence.

RPN Examples:

<table>
<thead>
<tr>
<th>Expression</th>
<th>RPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>11 + 22 + 33</td>
<td>11 22 + 33 +</td>
</tr>
<tr>
<td>(100 + 2) × 30</td>
<td>100 2 + 30 ×</td>
</tr>
<tr>
<td>(30 × 400) + (15 × 60)</td>
<td>30 400 × 15 60 × +</td>
</tr>
</tbody>
</table>

RPN expressions can be evaluated conveniently using a stack. The expression is read from left to right. When an operand is encountered, it is pushed on to the stack. When an operator is encountered, its operands are popped off the stack, the result is calculated and then pushed back on to the stack. The result of the evaluation of the expression should be the single value remaining on the stack after the entire expression has been processed.

Your subroutine should evaluate a single RPN expression stored as a NULL-terminated ASCII string in memory. You may assume that the string contains only operands, operators and spaces. Operators and operands are separated by a single space character (0x20). You may also assume that the RPN expression represented in the string is valid.

The operands are ASCII string representations of non-negative integers in decimal form. Your subroutine must support the following operators:
Postfix Evaluation

Infix Expression:

Any expression in the standard form like "2*3+5" is an Infix(expression).

Postfix Expression:

The Postfix(Preorder) form of the above expression is "23*5+".

Postfix Evaluation:

In normal algebra we use the infix notation like a+b*c. The corresponding postfix notation is abc*+.

The algorithm for the conversion is as follows:

- Scan the Postfix string from left to right.
- If the scanned character is an operand, add it to the stack. If the scanned character is an operator, there will be at least two operands in the stack.
- If the scanned character is an Operator, then we store the top most element of the stack(topstack) in a variable top. Pop the stack. Now evaluate top(stack(Operator)/top). Let the result of this operation be result. Pop the stack and push result into the stack.
- Repeat this step till all the characters are scanned.
- After all characters are scanned, we will have only one element in the stack. Return that stack.

Example:

Let us see how the above algorithm will be implemented using an example.

Postfix String: 23*4-

Initially the Stack is empty. Now, the first three characters scanned are 2, 3 and 4, which are operands. Thus they will be pushed into the stack in that order.

3
2
1
Stack

Next character scanned is "*", which is an operator. Thus, we pop the top two elements from the stack and perform the "*" operation with the two operands. The second operand will be the first element that is popped.

1
2*3=6
Expression

The value of the expression(2*3) that has been evaluated(6) is pushed into the stack.
Next character scanned is "+", which is an operator. Thus, we pop the top two elements from the stack and perform the "+" operation with the two operands. The second operand will be the first element that is popped.

The value of the expression (1+6) that has been evaluated (7) is pushed into the stack.

Next character scanned is "4", which is added to the stack.

Next character scanned is "-", which is an operator. Thus, we pop the top two elements from the stack and perform the "-" operation with the two operands. The second operand will be the first element that is popped.

The value of the expression (7-4) that has been evaluated (3) is pushed into the stack.
Now, since all the characters are scanned, the remaining element in the stack (there will be only one element in the stack) will be returned.

End result:

- Postfix String: 123*4-
- Result: 3