COL 216 Assignment 2: Evaluating Postfix Expressions

Ramneet Singh | Mustafa Chasmai
2019CS50445 | 2019CS10341

July 5, 2021

Design Characteristics:

1. Error handling in Input
    - If an invalid postfix expression is given, a relevant error message is displayed
    - Cases handled for when the input has characters other than numbers and operators.
    - Cases handled for when the input operators and operands do not match (the expression cannot be evaluated).
2. MIPS stack
    - The in-built memory stack of MIPS is used.
    - The specialised stack pointer variable provided by MIPS is used to perform the usual stack push and pop operations.
    - Depending on the original stack pointer, the size of the stack is calculated and checked whether the stack is empty or not.
3. Serial input
    - The characters of the expression are read sequentially.
    - This leads to no restrictions in the size of the input expression.
    - The size of the expression is bounded only by the MIPS memory stack size, which should be a bound for any other implementation design as well.

Design Choices:

1. We consider only those postfix expressions as valid, which have operands as integers in the range 0 - 9 and operators from +, -, or *.
2. The user is asked to input the expression once in a single prompt on the console.
3. The input is read character by character using multiple system calls.
4. Assuming 32 bit as the bound for values, each element of the stack was allotted 4 bytes of storage.

Approach of Solution:

A prompt is displayed on the console asking the user for the postfix expression to evaluate. The user has to type the sequence in one go, but the input is read by MIPS one character at a time.

The MIPS default stack is used for computations in this algorithm. as each character is read, the following steps are involved:

1. Read the first (next) character
2. If it is a number, push it onto the stack
3. If it an operator symbol:
   (a) Pop the two topmost elements of the stack
   (b) Perform the operation corresponding to this symbol
   (c) Push result of operation back onto the top of the stack
4. Repeat steps 1 to 3 till all characters are read and then pop the topmost element as the result.

The relevant errors are handled appropriately. Errors are flagged in step 1 if the input expression has symbols other than numbers and the +, - and * operators. Errors are flagged in step 3.a when the input expression is mismatched such that there are more operators than operands. Errors are flagged in step 4 when the input expression is mismatched such that there are more operands than operators (after the last pop, the stack is not empty).

Testing strategy:

1. **Random Testing:**
   Some random inputs are given to program and its output is checked with manual calculations. We have tried to include different operators and also to vary the length of the input expression that we provide.

No.	**Expression**	**Result**
1	98567++++	35
2	58+127*++	28
3	11-9*4-01-*	4
4	42*5-67+*	39
5	157++34*-	1
6	12+4*2-1+7*	77
7	54-9823+–*	-6
8	2394*-56+++	-20
9	12-9525+652+-*+-*	-11
10	74+9523*25+652+-*+-**	594

2. **Ultimate Test Case:**
   A single test case that includes all the major cases and hard computations encountered, along with a large number of operands in the expression.

Expression	**Result**
9882***4+74+9523*25+652+-*+-**74+9523*25+652+-*+-**+457++2*–	0

3. **Invalid inputs:**
   Some cases where inputs may not be as per specifications are identified and checked for errors or wrong outputs.

No.	**Expression**	**Result**
1		Error: Invalid postfix expression
2	abc+-1	Error: Invalid postfix expression
3	12/?	Error: Invalid postfix expression
4	0982+	Error: Invalid postfix expression
5	0+-**	Error: Invalid postfix expression
6	125.0+*	Error: Invalid postfix expression
7	-12*7*6*	Error: Invalid postfix expression