MIPS Postfix Evaluator

Aman Verma, Karan Aggarwal

2019CS50419, 2019CS10699

Design

The design is fairly simple and uses stack memory to store all the digits in a stack so that they can be popped as the operators appear.

Algorithm

We parse over the input postfix expression character by character and on the basis of the character being a number or operator we do the following:

1. Number: push to the stack
2. Operator: Take the top two elements of the stack, apply the operator on them, and push back the result to the stack.

However, this approach causes a significantly increased number of calls to the stack memory if implemented directly.

To reduce the number of calls to the stack, we store the topmost element of the stack in a register and the rest is stored in the stack memory. Let the current topmost element(in each iteration) be stored in $t1.

Terminology for clarity :

- Stack: includes the topmost element that is present in the register as well as the part present in the stack memory.
- Stack memory: the topmost element of the stack memory refers to the second element in the stack.

The modified procedure is as follows:

1. Number: Push the number currently in $t1 to the stack memory and store the currently parsed digit to $t1. This basically makes the currently parsed digit the topmost element of the stack and sends the previous top most digit to become the 2^{nd}, that is, to be the topmost in the stack memory.
2. Operator: We have the top most element of the stack in $t1 and getting one element from the stack memory gives us the 2nd element of the stack. Now the operator can be applied directly to these 2 numbers and the result can be stored back in $t1 since it is meant to contain the topmost number.
   This reduces the calls to memory from 3 to 1, thereby increasing efficiency significantly.

Error Detection

To deal with incorrect inputs we have the following checks:

- If empty string is entered then the stack is empty at the end of parsing. So an error is thrown.
- If at any point in the program, an operator appears and the number of elements in the stack is 1 or less then error is thrown.
- If at the end of parsing the string, the number of elements in the stack is greater than 1 then error is thrown as the postfix expression is incomplete.
- In case any character other than [0-9], *, +, - occurs in the postfix expression then an error is thrown.

Testing Strategy

There were several kinds of tests performed by us:

1. Random test cases generated using a C++ program (Testing/random_test_generator.cpp):
    - Large Cases (Testing/RandomCasesLarge.txt)
    - Medium Cases (Testing/RandomCasesMedium.txt)
    - Small Cases (Testing/RandomCasesSmall.txt)
2. Edge Cases:
    - Number of digits is more than expected
      Here, the stack has more than 1 element remaining after parsing user input. Example:
      Input: 123+6
      Answer: Invalid postfix string
    - Number of operators is more than expected
      Here, a pop operation is called on an empty stack and the error is handled. Example:
      Input: 45*+-6**
      Answer: Invalid postfix string
    - Empty string is entered. Example:
      Input:
      Answer: Invalid postfix string
    - A single digit is entered. Example:
      Input: 7
      Answer: 7
    - User input starts with an operator. Example:
      Input: *989
      Answer: Invalid postfix string