Stack

Time span: 2 lab days(4 hrs)

Problem: Development of arithmetic expressions evaluation system.

Description: An arithmetic expression combines arithmetic operators and operands in a valid way. Arithmetic operator i.e. **{+, -, *, /}** takes single or multiple operands according to their definitions. We use binary operators, takes two operands and single alphabet named variables(operands) in our expression evaluation system. Variable names can have single lowercase English alphabet only. The parenthesis, i.e. ***(*** and ***)*** are used to specify the precedence of sub-expression evaluation. In case of arithmetic expressions, the deeper enclosure by parenthesis indicates the higher precedence of evaluation.

We take input expression in the form of infix notation. For example,

x + y * (z / w)

is a an example input of typical arithmetic expression.

Solution:

1. Design and implement a Stack data structure (abstract data type) that can handle push/pop operations of single letter named variables of an arithmetic expression.

2. Convert the given infix expression to either postfix notation(position of operands followed by operator) or prefix notation(position of operator followed by operands). This removes precedence enclosed by the parenthesis(sub-expression).

For example, the postfix notation of the above example is:

x y z w / * +

3. Display the validity result, if the given expression is valid or not. Validity result could be invalid operands, name does not

belongs to the English lowercase alphabets, invalid binary operators and so on.

4. Read input value for the variables from the console and use them to evaluate the given arithmetic expression.

For example, in the above example x, y, z and w are the variables. So, we read 4 inputs for the respective variables as follows:

```
Input for x: ?
Input for y: ?
Input for z: ?
Input for w: ?
```

and finally we display the output after evaluating the resulted expression either in postfix or prefix notation.

For example: x = 5, y = 3, z = 100 and w = 4 then the evaluation output in postfix notation would be:

```
>> x y z w / * +
>> 5 3 100 4 / * +
>> 5 3 25 * +
>> 5 75 +
>> 80
```

Final result: 80.

***