Lecture

NOTE: FOR FURTHER DETAILS AND MORE COMPREHENSIVE STUDY, PLEASE SEE RECOMMENDED BOOKS OR INTERNET.

Stack

A ***Stack*** is a collection of objects inserted and removed according to the Last In First Out (***LIFO***) principle.

Postfix Expression Evaluation using Stack:

The expressions written in postfix form are evaluated faster compared to infix notation as parenthesis are not required in postfix. We have discussed infix to postfix conversion in previous lecture. In this lecture, we will discuss expression evaluation using stack.
Algorithm for postfix expression evaluation is as follows:

Algorithm:

1. Start
2. Read a ***given expression*** **left to right** and do following for every element.
   a. If ***number*** is encountered, ***push*** it onto STACK
   b. If ***operator*** is encountered, ***pop 2*** operands/values from STACK as follows:
      - 1st ***pop*** assign to **operand2**
      - 2nd ***pop*** assign to **operand1**
      - Evaluate the operation using the ***encountered operator***
      - ***push*** the result back onto STACK
3. Repeat ***3*** until all the elements have been read ***left to right***.
4. ***Pop*** the STACK one last time to get the result.
5. End

Example: Infix: (2+3*1–9), Postfix: (231*+9–)

1. Let the given expression be `'231*+9-'`, We scan elements one by one (left to right)
2. Scan `'2'`, it's a number, so push it onto stack. Stack contains `'2'`
3. Scan `'3'`, again a number, push it onto stack, stack now contains `'2 3'` (remember 1st number is in bottom and last one on top)
4. Scan `'1'`, again a number, push it onto stack, stack now contains `'2 3 1'`
5. 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'` onto stack. Stack now becomes `'2 3'`

6. Scan `'+'`, it's an operator, pop two operands from stack, apply the `+` operator on operands, we get `2+3` which results in `5`. We push the result `'5'` onto stack. Stack now becomes `'5'`
7. Scan `'9'`, it's a number, we push it onto stack. Stack now becomes `'5 9'`
8. 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'` onto stack. Stack now becomes `'-4'`
9. There are no more elements to scan, we return the top element from stack (which is the only element left in stack).

Prefix Expression Evaluation using Stack:

As we discussed infix to prefix conversion earlier we will now discuss the prefix expression evaluation using stack. Prefix expression is evaluated basically the same way postfix expression but with few minor steps changed.
Following is algorithm for evaluation postfix expressions.

Algorithm:

1. Start
2. Read a ***given expression*** **right to left** and do following for every element.
   c. If ***number*** is encountered, ***push*** it onto STACK
   d. If ***operator*** is encountered, ***pop 2*** operands/values from STACK as follows:
      - 1st ***pop*** assign to **operand1**
      - 2nd ***pop*** assign to **operand2**
      - Evaluate the operation using the ***encountered operator***
      - ***push*** the result back onto STACK
3. Repeat ***3*** until all the elements have been read ***right to left***.
4. ***Pop*** the STACK one last time to get the result.
5. End

Example: Infix: (2+3*1–9), Prefix: (–+2*319)

1. Let the given expression be `'-+2*319'`, We scan elements one by one (right to left)
2. Scan `'9'`, it's a number, so push it onto stack. Stack contains `'9'`
3. Scan `'1'`, again a number, push it onto stack, stack now contains `'9 1'` (remember 1st number is in bottom and last one on top)
4. Scan `'3'`, again a number, push it onto stack, stack now contains `'9 1 3'`
5. 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'` onto stack. Stack now becomes `'9 3'`
6. Scan `'2'`, it's a number, push it onto stack, stack now contains `'9 3 2'`

7. Scan `'+'`, it's an operator, pop two operands from stack, apply the **+** operator on operands, we get **2+3** which results in **5**. We push the result `'5'` onto stack. Stack now becomes `'9 5'`
8. 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'`
9. There are no more elements to scan, we return the top element from stack (which is the only element left in stack).

Using the given algorithms we can evaluate any prefix or postfix expression.