Arithmetic Expressions

- An expression is a string of symbols.
- Arithmetic expressions are made up of variable names, binary operators, and brackets. But in actual computer languages, there are many other things such as powers(**), unary minus(-a), numbers(22/7*3.12a) and things like function(a=find(a,b)+c) and array references may be present.
- We are going to consider the expressions with variables(a-z), digits, binary operators(+,-,*,/) and brackets([-left & ]-right).
- Example of some arithmetic expression:
  - a+b-c
  - a+b+c*d
  - (a+b)*(c-d)

Types of Expression:

- An expression can be in 3 forms:
  1. Infix Expression
  2. Prefix Expression
  3. Postfix Expression
- Infix, prefix, and postfix notations are different ways of writing expressions.
- In the 3 ways, the operands occur in the same order but the operators have to be moved.
- We are using infix type of expression in our daily life, but the computer uses postfix or prefix type of expression.

1. Infix Notation:

- Operators are written in between their operands.
- This is used in our common mathematical expressions.
- The operations(order of evaluation) are performed from left to right, and it obeys precedence rules (multiplication and division are performed before addition and subtraction).
- Brackets can be used to change the order of evaluation.
- Examples:
  - (a) A+B  (b) X*(Y+Z)

2. Prefix Notation (Polish notation)
- Operators are written before their operands
- order of evaluation from right to left.
- example
  (a) +AB  (b)*X+YZ

3. Postfix Notation (Reverse Polish notation)

- Operators are written after their operands
- The order of evaluation of operators is always from left to right
- brackets cannot be used to change the order of evaluation.
- Example
  (a) AB+, (b) XYZ+*

Operator Precedence:

- In the table the precedence is decreasing downwardly
- if there are two operators with same precedence then computer start to solve the expression from left to right

<table>
<thead>
<tr>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>()</td>
</tr>
<tr>
<td>* , / , %</td>
</tr>
<tr>
<td>+ , -</td>
</tr>
</tbody>
</table>

Arithmetic Expression Evaluation:

- An important application of stacks is parsing, ie a compiler must evaluate arithmetic expressions written using infix notation.
- The problem of parsing infix expression can be break in to 2 stages
  1. Infix to Postfix Conversion
  2. Evaluating a Postfix expression
- Converting an infix expression in to postfix expression and evaluating a Postfix expression is a easier problem than directly evaluating Infix expression

Source:

http://datastructuresprogramming.blogspot.in/2010/02/expression-evaluation.html