Expression evaluation

Postfix also known as Reverse Polish Notation (or RPN), is a notational system where the operation/function follows the arguments. For example, "1 2 add" would be postfix notation for adding the numbers 1 and 2.

Programming languages use either

* prefix notation " add 1 2” or “+ 1 2 “.
* infix notation "1 add 2" or "1 + 2".
* Postfix notation “1 2 add” or “1 2 +”.

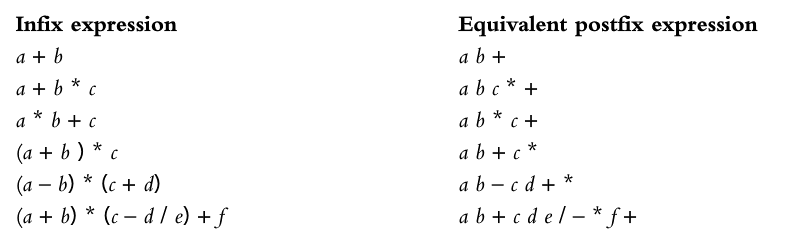
Prefix and infix are more familiar to most people, as they are the standard notations used for arithmetic and algebra. Many people wonder why anyone would use this "weird" postfix notation.

The answer is that it is useful, especially for programming, because it clearly shows the order in which operations are performed, and because it disambiguates operator groupings. For example, look at this postfix expression: 1 2 + 3 \* 6 + 2 3 + /

This means "take 1 and 2, add them, take 3 and multiply, take 6 and add, take 2 and 3, add them, and divide".

In contrast, the equivalent expression in Infix Notation is: (((1 + 2) \* 3) + 6) / (2 + 3)

This may seem more familiar, but note the need for parentheses to control the order of evaluation. The prefix notation would be: (/ (+ (\* (+ 1 2) 3) 6) (+ 2 3))



**Algorithm: Infix to Postfix notation:**

* If we get an operand then write it down
* If operator
  + stack is empty, push on stack
  + stack not empty
    - if current operator is higher in precedence than what is on top of the stack then push
    - if current operator is lower or equal in precedence than what is on top of the stack then pop and display what is on the stack. Repeat process with the hot potatoe operator to resolve it.
* If open parenthesis, then push on stack
* If close parenthesis, pop and **display** everything up to the open parenthesis. Pop the open parenthesis. **Do not display** open parenthesis.
* If nothing else to process, pop and display everything

The operands will appear in the same order. The operators will appear in the order that they are supposed to be evaluated.

Summary: Push (Empty, operator higher, open parenthesis)

Pop (operator lower or equal, close parenthesis(pop to open parenthesis (inclusive)) , nothing else to process)

Display (operands, what is popped)

Do not display parentheses

Precedence from highest to lowest

^

\*/

+-

(

=

Optional: Display can be exchanged with enqueue for later processing of postfix notation.

**Example 1**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Expression to be evaluated:** a+b\*(c-d) | | | | |
| **What is being processed** | **What is done** | **Why is it done** | **Contents of stack** | **What is displayed** |
| Operand: a | Display | Operands are always displayed | Empty | **a** |
| Operator: + | Push | Stack is empty | **+** |  |
| Operand: b | Display | Operands are always displayed |  | a **b** |
| Operator: \* | Push | Higher precedence than previous operator, +, on stack | + **\*** |  |
| Operator: ( | Push | ( are always pushed on stack | + \* **(** |  |
| Operand: c | Display | Operands are always displayed |  | a b **c** |
| Operator: - | Push | Higher precedence than previous operator, (, on the stack | + \* ( **-** |  |
| Operand: d | Display | Operands are always displayed |  | a b c **d** |
| Operator: ) | Pop and display | When right parenthesis, pop and display everything up to the left parenthesis. Pop the left parenthesis, but don’t display it | + \* | a b c d **-** |
| Nothing else is left | Pop and display | The expression has nothing else left so pop and display everything |  | a b c d **- \* +** |

**Example 2**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Expression to be evaluated: f=a\*c ^b/p-q\*g^(h-b)** | | | | |
| **What is being processed** | **What is done** | **Why is it done** | **Contents of stack** | **What is displayed** |
| Operand: f | Display | Operands are always displayed | Empty | **f** |
| Operator: = | Push | Stack is empty | **=** |  |
| Operand: a | Display | Operands are always displayed |  | f a |
| Operator: \* | Push | Higher precedence than previous operator, =, on stack | = \* |  |
| Operand: c | Display | Operands are always displayed |  | f a c |
| Operator: ^ | Push | Higher precedence than previous operator, \*, on stack | = \* ^ |  |
| Operand: b | Display | Operands are always displayed |  | f a c b |
| Operator: / | Pop and display | lower precedence than previous operator, =, on stack | = \* | f a c b ^ |
|  | Pop and display | Same precedence as operator on top, \* | = | f a c b ^ \* |
|  | push | Higher precedence than operator on top, = | = / |  |
| Operand: p | Display | Operands are always displayed |  | f a c b ^ \* p |
| Operator: - | Pop and display | lower precedence than operator on top, / | = | facb^\*p/ |
|  | push | Higher precedence than operator on top, = | = - |  |
| Operand: q | Display | Operands are always displayed |  | facb^\*p/q |
| Operator: \* | Push | Higher precedence than top item, -, on stack | = - \* |  |
| Operand: g | Display | Operands are always displayed |  | facb^\*p/qg |
| Operator: ^ | Push | Higher precedence than top item, \*, on stack | = -\* ^ |  |
| Operator: ( | Push | ( are always pushed on stack | = -\* ^ **(** |  |
| Operand: h | Display | Operands are always displayed |  | facb^\*p/qgh |
| Operator: - | Push | Higher precedence than top item, (, on stack | = -\* ^ ( - |  |
| Operand: b | Display | Operands are always displayed |  | facb^\*p/qghb |
| Operator: ) | Pop and display | When right parenthesis, pop and display everything up to the left parenthesis. Pop the left parenthesis, but don’t display it | = -\* ^ | facb^\*p/qghb- |
| Nothing else is left | Pop and display | The expression has nothing else left so pop and display everything |  | facb^\*p/qghb-^\*-= |

**Algorithm: Evaluating Postfix expression**

Operators:

The symbol is one of the following: +, -, \*,/ or =.

If the symbol is +, -, \*, or /, the symbol is an operator and so we must evaluate it. Because an operator requires two operands, the stack must have at least two elements; otherwise, the expression has an error.

If the symbol is =, the expression ends and we must print the answer. At this step, the stack must contain exactly one element; otherwise, the expression has an error.

The symbol we read is something other than +, -, \*,/ or =.

In this case, the expression contains an illegal operator.

Operands:

Push on the stack

Summary:

when operand, push (operand)

When operator, pop (2 operands), evaluate and push result

If operator is =, pop result

**Example 1 (evaluation)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Infix : (1+2)-(3+4) \* 5**  **Postfix: 12+ 34 + 5\*- This is what is being processed** | | | | |
| **What is being processed** | **What is done** | **Why is it done** | **Contents of stack** | **Result** |
| Operand: 1 | push | Operands are always pushed | 1 |  |
| Operand: 2 | push | Operands are always pushed | 1 2 |  |
| Operator: + | Pop two items, calculate and push | With operator, pop two items | 3 |  |
| Operand: 3 | push | Operands are always pushed | 3 3 |  |
| Operand: 4 | push | Operands are always pushed | 3 3 4 |  |
| Operator: + | Pop two items, calculate and push | With operator, pop two items | 3 7 |  |
| Operand: 5 | push | Operands are always pushed | 3 7 5 |  |
| Operator: \* | Pop two items, calculate and push | With operator, pop two items | 3 35 |  |
| Operator: - | Pop two items, calculate and push | With operator, pop two items | -32 |  |

**Example 2 (evaluation)**

