* While there are tokens to be read:
* Read a [token](http://en.wikipedia.org/wiki/Token_(parser)).
* If the token is a number, then add it to the output queue.
* If the token is a [function](http://en.wikipedia.org/wiki/Function_(mathematics)) token, then push it onto the stack.
* If the token is a function argument separator (e.g., a bracket):
* Until the token at the top of the stack is a left parenthesis, pop operators off the stack onto the output queue. If no left parentheses are encountered, either the separator was misplaced or parentheses were mismatched.
* If the token is an operator, o1, then:
* while there is an operator token, o2, at the top of the stack, and

either o1 is [left-associative](http://en.wikipedia.org/wiki/Operator_associativity) and its [precedence](http://en.wikipedia.org/wiki/Order_of_operations) is equal to that of o2,

or o1 has precedence less than that of o2,

pop o2 off the stack, onto the output queue;

* push o1 onto the stack.
* If the token is a left parenthesis, then push it onto the stack.
* If the token is a right parenthesis:
* Until the token at the top of the stack is a left parenthesis, pop operators off the stack onto the output queue.
* Pop the left parenthesis from the stack, but not onto the output queue.
* If the token at the top of the stack is a function token, pop it onto the output queue.
* If the stack runs out without finding a left parenthesis, then there are mismatched parentheses.
* When there are no more tokens to read:
* While there are still operator tokens in the stack:
* If the operator token on the top of the stack is a parenthesis, then there are mismatched parentheses.
* Pop the operator onto the output queue.
* Exit.

In reverse Polish notation the [operators](http://en.wikipedia.org/wiki/Operation_(mathematics)) follow their [operands](http://en.wikipedia.org/wiki/Operands); for instance, to add 3 and 4, one would write "3 4 +" rather than "3 + 4". If there are multiple operations, the operator is given immediately after its second operand; so the expression written "3 - 4 + 5" in conventional notation would be written "3 4 - 5 +" in RPN: first subtract 4 from 3, then add 5 to that. An advantage of RPN is that it obviates the need for parentheses that are required by infix. While "3 - 4 \* 5" can also be written "3 - (4 \* 5)", that means something quite different from "(3 - 4) \* 5". In postfix, the former could be written "3 4 5 \* -", which unambiguously means "3 (4 5 \*) -" which reduces to "3 20 -"; the latter could be written "3 4 - 5 \*" (or 5 3 4 - \*, if you wish to keep similar formatting), which unambiguously means "(3 4 -) 5 \*".

Despite the name, reverse Polish notation is not exactly the reverse of Polish notation, for the operands of non-[commutative](http://en.wikipedia.org/wiki/Commutative) operations are still written in the conventional order (e.g. "/ 6 3" in Polish notation and "6 3 /" in reverse Polish both evaluate to 2, whereas "3 6 /" in reverse Polish notation would evaluate to ½).

The algorithm for evaluating any postfix expression is fairly straightforward:

* While there are input tokens left
  + Read the next token from input.
  + If the token is a value
    - Push it onto the stack.
  + Otherwise, the token is an operator (operator here includes both operators and functions).
    - It is known [*a priori*](http://en.wikipedia.org/wiki/A_priori_and_a_posteriori) that the operator takes **n** arguments.
    - If there are fewer than **n** values on the stack
      * **(Error)** The user has not input sufficient values in the expression.
    - Else, Pop the top **n** values from the stack.
    - Evaluate the operator, with the values as arguments.
    - Push the returned results, if any, back onto the stack.
* If there is only one value in the stack
  + That value is the result of the calculation.
* Otherwise, there are more values in the stack
  + **(Error)** The user input has too many values.

### Example[[edit](http://en.wikipedia.org/w/index.php?title=Reverse_Polish_notation&action=edit&section=4)]

The infix expression "5 + ((1 + 2) \* 4) − 3" can be written down like this in RPN:

5 1 2 + 4 \* + 3 -

The expression is evaluated left-to-right, with the inputs interpreted as shown in the following table (the *Stack* is the list of values the algorithm is "keeping track of" after the*Operation* given in the middle column has taken place):

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Operation** | **Stack** | **Comment** |
| 5 | Push value | 5 |  |
| 1 | Push value | 1 5 |  |
| 2 | Push value | 2 1 5 |  |
| + | Add | 3 5 | Pop two values (1, 2) and push result (3) |
| 4 | Push value | 4 3 5 |  |
| \* | Multiply | 12 5 | Pop two values (3, 4) and push result (12) |
| + | Add | 17 | Pop two values (5, 12) and push result (17) |
| 3 | Push value | 3 17 |  |
| − | Subtract | 14 | Pop two values (17, 3) and push result (14) |
|  | Result | (14) |  |

When a computation is finished, its result remains as the top (and only) value in the stack; in this case, 14.

The above example could be rewritten by following the "chain calculation" method described by [HP](http://en.wikipedia.org/wiki/HP) for their series of RPN calculators:[[7]](http://en.wikipedia.org/wiki/Reverse_Polish_notation#cite_note-7)

As was demonstrated in the Algebraic mode, it is usually easier (fewer keystrokes) in working a problem like this to begin with the arithmetic operations inside the parentheses first.

1 2 + 4 \* 5 + 3 −

## Converting from infix notation[[edit](http://en.wikipedia.org/w/index.php?title=Reverse_Polish_notation&action=edit&section=5)]

*Main article:*[*Shunting-yard algorithm*](http://en.wikipedia.org/wiki/Shunting-yard_algorithm)

[Edsger Dijkstra](http://en.wikipedia.org/wiki/Edsger_Dijkstra) invented the [shunting-yard algorithm](http://en.wikipedia.org/wiki/Shunting-yard_algorithm) to convert infix expressions to postfix (RPN), so named because its operation resembles that of a [railroad shunting yard](http://en.wikipedia.org/wiki/Classification_yard).

There are other ways of producing postfix expressions from infix notation. Most [operator-precedence parsers](http://en.wikipedia.org/wiki/Operator-precedence_parser) can be modified to produce postfix expressions; in particular, once an[abstract syntax tree](http://en.wikipedia.org/wiki/Abstract_syntax_tree) has been constructed, the corresponding postfix expression is given by a simple [post-order traversal](http://en.wikipedia.org/wiki/Post-order_traversal) of that tree.