

The background features a dark blue gradient with numerous vertical and diagonal streaks of bright blue light, creating a sense of depth and movement. A large, irregular black brushstroke is positioned on the right side, serving as a backdrop for the text.

Reverse Polish Notation

MARTZEL BASTE

Reverse Polish Notation (RPN)

- was devised as a method of simplifying mathematical expressions.
- It predates modern computers.
- Early program translators converted expressions to RPN for evaluation.

There are different types of expression formats:

Pre A In B Post

Prefix expression

• $* + a b - c d$

Infix expression

• $(a+b) * (c-d)$

Postfix expression

• $a b + c d - *$

Infix Evaluation $2+3*5$

PMDAS

$$2+3*5$$

- $= 2+15$
- $= 17$

$$(2+3)*5$$

- $= 5*5$
- $= 25$

Infix notation requires Parentheses.

Prefix Evaluation

$$+ 2 * 3 5 =$$

Handwritten annotations: A bracket under $2 + 3$ with the result 5 written below it. An arrow points from this 5 to the $*$ operator, and another arrow points from the 5 to the final 5 operand.

$$= + 2 \underline{* 3 5}$$

$$= \underline{+ 2 15} = 17$$

$$* + 2 3 5 =$$

Handwritten annotations: A bracket under $2 + 3$ with the result 5 written below it. An arrow points from this 5 to the $*$ operator, and another arrow points from the 5 to the final 5 operand.

$$= * \underline{+ 2 3 5}$$

$$= \underline{* 5 5} = 25$$

No parentheses
needed!

Postfix Evaluation

$2\ 3\ 5\ * \ + \ =$



$= 2\ \underline{3\ 5\ *} \ +$

$= \underline{2\ 15} \ + \ = 17$

$(2\ 3) \ + \ 5$

$2\ 3 \ + \ 5\ * \ =$



$= \underline{2\ 3} \ + \ 5\ *$

$= \underline{5\ 5} \ * \ = 25$

No parentheses
needed here either!



Fully Parenthesized Expression (FPE)

- A FPE has exactly one set of Parentheses enclosing each operator and its operands.
- Which is fully parenthesized?

☐ $(A + B) * C$

☒ $((A + B) * C)$ ✓

☐ $((A + B) * (C))$

$\{(23) * (3+4)+5\}$

Infix to Prefix Conversion

Move each operator to the **left** of its operands & remove the parentheses:

$$\begin{array}{c} \times ((A+B) * (C+D)) \\ \text{---} \\ * + A B + C D \end{array}$$

$$\begin{array}{c} \leftarrow (((A+B) * C) - ((D+E) / F)) \\ \text{---} \\ \text{---} \quad * + A B C \quad \cdot \quad / + D E F \\ \text{---} \\ \text{---} \quad * + A B C / + D E F \end{array}$$

Infix to ~~Post fix~~ Prefix Conversion

Move each operator to the **right** of its operands & remove the parentheses:

$$((A + B) * (C + D))$$

Diagram illustrating the conversion of the infix expression $((A + B) * (C + D))$ to postfix notation. The operators are moved to the right of their operands, resulting in the postfix expression $A B + C D + *$.

$$(((A + B) * C) - ((D + E) / F))$$


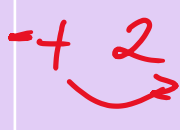










Diagram illustrating the conversion of the infix expression $(((A + B) * C) - ((D + E) / F))$ to postfix notation. The operators are moved to the right of their operands, resulting in the postfix expression $A B + C * D E + F / -$.

- ⇒ Operand order does not change!
- ⇒ Operators are in order of evaluation!

Prefix to Postfix Conversion (Vice Versa)

Convert to Infix, then to a certain notation.

$- + 4 9 8 \rightarrow$	<u>Infix</u> $4 + 9 - 8 \rightarrow$ $13 - 8 = 5$	<u>Postfix</u> $4 9 + 8 - \rightarrow$ $13 - 8 = 5$
$* + 2 3 - 4 8 \rightarrow$	$2 + 3 * 4 - 8 \rightarrow$	$2 3 + 4 8 - *$

Infix	Prefix 	Postfix 
$2 + 3 - 4 = 1$ 	$+ 2 3 - 4 = 1$ 	$2 3 + 4 - = 1$ 
$2 + (3 - 4) = 1$	$+ 2 - 3 4$	$2 3 4 - +$
$2 + 3 * 4 = 12 + 2 = 14$ 	$+ 2 * 3 4$ $2 + 12 = 14$	$2 3 4 * + = 14$ 
$(2 + 3) * 4$  $5 * 4 = 20$	$* + 2 3 4 = 20$ 	$2 3 + 4 * = 20$ 
$(2 - 3) * (4 + 5)$  	$* - 2 3 + 4 5$	$2 3 - 4 5 + *$
$2 + 8 * 5 / 10$  L to R $40 / 10 = 4$ $2 + 4 = 6$	$+ 2 / * 8 5 10$   0 4	$2 8 5 * 10 / =$   40 10 $4 = 6 \checkmark$

Advantages

- RPN expressions do not need brackets and there is no need for precedence of operators
- RPN is simpler for a machine to evaluate
- There is no need for backtracking in evaluation as the operators appear in the order required for computation and can be evaluated from left to right.

Implementation

- Stack
- Trees – Expression Trees