* To get rid of parantheses we have 3 stacks: prefix, expression and "no brackets expression". Initially prefix stack and "no brackets expression" are empty and expression stack is original expression. Top of expression stack is left most token.

For example:

$$(1+2-(3+4/(2-5)*6))$$

$$\underbrace{(1+2-(3+4/(2}_{\text{prefix}}\ \underset{\substack{\uparrow\\ \text{token}}}{-}\ \underbrace{5)*6))}_{\text{expression}}$$

top of prefix stack

top of expression stack

We go in expression from left to right and skip all tokens (and push them to prefix stack) until we hit ")".

We then pop back from prefix stack all tokens (and push them to "no brackets expression" stack) until we hit "(".

we then evaluate "no brackets expression" and push result back to prefix and clear nbe.

expression = "(1+2-(3+4/(2-5)*6))"
prefix = " "
nbe = " "
token = " "

prefix = "(1+2-(3+4/(2-5"
token = ")"
expression = "*6))"
nbe = " "

prefix = "(1+2-(3+4/"
token = "("
expression = "*6))"
nbe = "2-5"

Evaluate (nbe) = "-3"
clear nbe
push "-3" to prefix

prefix = "(1+2-(3+4/-3"

expression = "*6))"
nbe = ""

we continue until we reach end of expression.
Resulting prefix will be expression without parantheses.
We then evaluate prefix.

*Evaluate expression without paranthesis:
* we have prefix and expression stacks.
* Initially prefix stack is empty and expression stack is equal to expression without paranthesis.
* If there are 4 operations we evaluate

expression 4 times in order of operation precedence.

When the evaluation is completed, expression is empty and prefix has no occurrences of "operation" operand.

* Evaluation works as follows:
  We pop from expression stack and push to prefix stack until we hit "operation" token.
  We then pop left operand from prefix stack and right operand from expression stack.
  We calculate result and push it to prefix stack.
  We until we reach end of expression.

For example, to evaluate / :

prefix = " "

expression = "1+2-3 / 5 *6 / 7 "

prefix = " 1+2-3"

expression = "/5 *6 /7 "

prefix = " 1+2-"

op1 = 3 result = 3/5 = 0

op2 = 5

expression = " *6 /7"

prefix = " 1+2-0 "

expression = " *6 /7 "

prefix = " 1+2-0 *6"

expression = " /7"

prefix = " 1+2-0 * "

op1 = 6 result = 6/7 = 0

op2 = 7 expression = ""

```
prefix = "1+2-0 * 0"

expression = ""
```