Shunting-Yard Algorithm (Explaination)

Condition / Loop		postfix	postfix = postfix + c	infix= a.(b.b)*.a for c in infix	stack = stack + c	stack (oprators/functions)	
1	a	a	save	a			a saved into postfix
				.	save	.	dot saved into stack
				(	save	(	Second delete open (from stack
2	ab	b	save	b			b saved into postfix
				.	save	.	First save dot Into postfix
3	abb	b		b			and then delete from stack
			save	)	Delete	Delete from stack until (stack[:-1] != (	
4	abb.	.	save				
5	abb.*	*	save	*		prec[stack[-1]] >= prec[c]	Condition = False (dot > *)
6	abb.*.	.	save	.		prec[stack[-1]] >= prec[c]	Condition = True (dot >= dot)
7	abb.*.a	a	save	a			

Postfix abb.*.a

Sudo Code https://en.wikipedia.org/wiki/Shunting-yard_algorithm

```
while there are tokens to be read:
    read a infix.
    if the c is a number, then:
        push it to the postfix
    else if the c is a function then:
        push it onto the stack
    else if the c is an operator then:
        while ((there is an operator at the top of the operator stack)
              and ((the operator at the top of the operator stack has greater precedence)
                  or (the operator at the top of the operator stack has equal precedence and the c is left associative))
              and (the operator at the top of the operator stack is not a left parenthesis)):
            pop operators from the operator stack onto the output queue.
        push it onto the operator stack.
    else if the token is a left parenthesis (i.e. "("), then:
        push it onto the operator stack.
    else if the token is a right parenthesis (i.e. ")"), then:
        while the operator at the top of the operator stack is not a left parenthesis:
            pop the operator from the operator stack onto the output queue.
        /* If the stack runs out without finding a left parenthesis, then there are mismatched parentheses. */
        if there is a left parenthesis at the top of the operator stack, then:
            pop the operator from the operator stack and discard it
        if there is a function token at the top of the operator stack, then:
            pop the function from the operator stack onto the output queue.
/* After while loop, if operator stack not null, pop everything to output queue */
if there are no more tokens to read then:
    while there are still operator tokens on the stack:
        /* If the operator token on the top of the stack is a parenthesis, then there are mismatched parentheses. */
        pop the operator from the operator stack onto the output queue.
exit.
```