Infix to Postfix:

Algorithm:
1. Create an empty stack `opstack` and list `postfix`.
2. Scan expression from left to right:
   2.1. If the token is an operand, append it to `postfix`.
   2.2. If the token is a `(`, push it to `opstack`.
   2.3. If the token is a `)`, pop until the corresponding `(` is popped and append to `postfix` list.
   2.4. If the token is an operator:
      2.4.1. Remove operator with higher or equal precedence.
      2.4.2. Push operator on the `opstack`.
3. When the expression is complete, pop from stack & append to postfix list.

Precedence Rules:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>^</td>
<td>(Power)</td>
</tr>
<tr>
<td>2</td>
<td>*, /</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>+, -</td>
<td>C</td>
</tr>
</tbody>
</table>

Example: \[(A \% D)^C \times E + X - Y \times (A^B)\]

1. Scan `(`, it's a token (line 2.2), push it to `opstack`.
2. Scan `A`, it's an operand, append it to `postfix` list:
   \[\text{postfix} = \text{'A'}, \quad \text{opstack} = \text{['C']}\]
3. Scan `\%`, it's an operator, there's no operators in the `opstack` (line 2.4.2), push operator to `opstack` (line 2.4.4):
   \[\text{opstack} = \text{[['C'], ['%']}}, \quad \text{postfix} = \text{'A'}\]
4. Scan `D`, it's an operand, append to `postfix` list:
   \[\text{postfix} = \text{'AD'}, \quad \text{opstack} = \text{[['C'], ['%'],['D']}}\]
5. Scan `)`, pop until corresponding `(` is popped (line 2.3), there is no operator in the `opstack` (line 2.4.1):
   \[\text{opstack} = \text{[}}, \quad \text{postfix} = \text{'AD\%'}\]
6. Scan `^`, it's an operator, I can push to the `opstack` (line 2.4.4):
   \[\text{opstack} = \text{[}, \quad \text{postfix} = \text{'AD\%'}\]
7. Scan `C`, it's an operand, append it to `postfix` list:
   \[\text{postfix} = \text{'AD\%C'}, \quad \text{opstack} = \text{[}}\]
8. Scan `\times`, it's an operator, pop until corresponding `(` is popped (line 2.3), there is no operator in the `opstack` (line 2.4.1):
   \[\text{opstack} = \text{[}}, \quad \text{postfix} = \text{'AD\%C'}\]
9. Scan `E`, it's an operand, append to `postfix` list:
   \[\text{postfix} = \text{'AD\%C E'}, \quad \text{opstack} = \text{[}}\]
10. Scan `-`, it's an operator, pop until corresponding `(` is popped (line 2.3), there is no operator in the `opstack` (line 2.4.1):
    \[\text{opstack} = \text{[}}, \quad \text{postfix} = \text{'AD\%C E -'}\]
11. Scan `Y`, it's an operand, append to `postfix` list:
    \[\text{postfix} = \text{'AD\%C E - Y'}, \quad \text{opstack} = \text{[}}\]
12. Scan `\times`, it's an operator, pop until corresponding `(` is popped (line 2.3), there is no operator in the `opstack` (line 2.4.1):
    \[\text{opstack} = \text{[}}, \quad \text{postfix} = \text{'AD\%C E - Y \times'}\]
13. Scan `A`, it's an operand, append to `postfix` list:
    \[\text{postfix} = \text{'AD\%C E - Y \times A'}, \quad \text{opstack} = \text{[}}\]
14. Scan `^`, it's an operator, I can push to the `opstack` (line 2.4.4):
    \[\text{opstack} = \text{[}, \quad \text{postfix} = \text{'AD\%C E - Y \times A^'}\]
15. Scan `B`, it's an operand, append to `postfix` list:
    \[\text{postfix} = \text{'AD\%C E - Y \times A^B'}, \quad \text{opstack} = \text{[}}\]
16. Scan `)`, it's a token (line 2.3), pop until corresponding `(` is popped (line 2.3), there is no operator in the `opstack` (line 2.4.1):
    \[\text{opstack} = \text{[}}, \quad \text{postfix} = \text{'AD\%C E - Y \times A^B'}\]
17. Scan `+`, it's an operator, pop until corresponding `(` is popped (line 2.3), there is no operator in the `opstack` (line 2.4.1):
    \[\text{opstack} = \text{[}}, \quad \text{postfix} = \text{'AD\%C E - Y \times A^B +'}\]
18. Scan `X`, it's an operand, append to `postfix` list:
    \[\text{postfix} = \text{'AD\%C E - Y \times A^B + X'}, \quad \text{opstack} = \text{[}}\]
19. Scan `)`, it's a token (line 2.3), pop until corresponding `(` is popped (line 2.3), there is no operator in the `opstack` (line 2.4.1):
    \[\text{opstack} = \text{[}}, \quad \text{postfix} = \text{'AD\%C E - Y \times A^B + X -'}\]
3. Scan `/`, it's an operator, pop all operators with same or higher precedence (line 2.4.1) 
`^` had higher precedence than `/`, so `/` will be popped from the stack and appended to postfix list.

```
postfix = 'AD%C^'
opstack = [ ]
```

The new operator is pushed to stack (line 2.4.2)

```
opstack = [ ]
```

4. Scan `E`, it's an operand, append it to postfix list (line 2.1)

```
postfix = 'AD%C^E'
opstack = [ ]
```

9. Scan `*`, it's an operator, `*` is popped from opstack because it has higher precedence (see B) and appended to postfix list (line 2.4.1) `*` is pushed to opstack (line 2.4.3)

```
postfix = 'AD%C^E*

opstack = [ ][ ]
```

10. Scan `+`, it's an operator, append it to postfix list (line 2.1)

```
postfix = 'AD%C^E+X'
opstack = [ ]
```

12. Scan `^`, it's an operator, pop `^` from opstack because it has same precedence (see C) and append it to postfix list (line 2.4.1)

```
postfix = 'AD%C^E*X+
opstack = [ ]
```

13. Scan `/`, it's an operator, append it to postfix list (line 2.4.1)

```
postfix = 'AD%C^E*X+Y'
opstack = [ ]
```

14. Scan `*`, it's an operator, pop all operators with same or higher precedence (line 2.4.1), `+` has lower precedence, so it won't be popped.

Then, push `*` to opstack

```
postfix = 'AD%C^E*X+Y'
opstack = [ ]
```

15. Scan `C`, push `1` to opstack (line 9.3)

```
postfix = 'AD%C^E*X+Y'
opstack = [ ]
```

16. Scan `A`, it's an operand (line 2.1)

```
postfix = 'AD%C^E*X+YA'
opstack = [ ]
```

17. Scan `^`, it's an operator, the stack has `(` as last element, so we can push the operator to opstack.

```
postfix = 'AD%C^E*X+YA`
opstack = [ ]
```

18. Scan `/`, it's an operator, (line 2.4.2) so, append to postfix list

```
postfix = 'AD%C^E*X+YA`
opstack = [ ]
```

19. Scan `)`, pop until corresponding `(` (line 9.3)

```
postfix = 'AD%C^E*X+YA^n
opstack = [ ]
```

(a) Scan `,`, pop until corresponding `(` (line 9.3)

```
postfix = 'AD%C^E*X+YA^n
opstack = [ ]
```

(b) End of expression (line 9.7), pop all 

append to list.

```
distri = 'AD%C^E*X+YAB'```