Postfix to Infix:

Algorithm:
1. Create an empty stack `odstack`
2. Scan the expression from left to right
   a. if the token is an operand, push on `odstack`
   b. if the token is an operator (op):
      1. pop from the stack: op2
      2. pop from the stack: op1
      3. compute result: `res = op1 [op] op2`
      4. push the result in `odstack`
3. When the expression is complete, pop result from `odstack`

Example:

```
AD% C ^ E * X ± Y A B ^ * =
```

1. Scan `A`, it's an operand. Line 2.1 of the algorithm.
2. Scan `D`, it's an operand. Line 2.1 of the algorithm.
3. Scan `%`, it's an operator. Line 2.2 of the algorithm.
   - (Line 2.2.1) `op2 = D` (pop 2)
   - (Line 2.2.2) `op1 = A` (pop 1)
   - (Line 2.2.3) `res1 = op1 % op2` (res1)
   - (Line 2.2.4) push res1 in stack.
4. Scan `C`, it's an operand. Line 2.1 of the algorithm.
5. Scan `Y`, it's an operator. Line 2.2 of the algorithm.
   - (Line 2.2.1) `op2 = C` (pop 1)
   - (Line 2.2.2) `op1 = res1` (pop 1)
   - (Line 2.2.3) `res2 = res1 ^ C` (res2)
   - (Line 2.2.4) push res2 in stack.
6. Scan `E`, it's an operand. Line 2.1 of the algorithm.
7. Scan `*`, it's an operator. Line 2.2 of the algorithm.
   - (Line 2.2.1) `op2 = E` (pop 1)
   - (Line 2.2.2) `op1 = res2` (pop 1)
   - (Line 2.2.3) `res3 = res2 * E` (res3)
   - (Line 2.2.4) push res3 in stack.
8. Scan `X`, it's an operand. Line 2.1 of the algorithm.
9. Scan `+`, it's an operator. Line 2.2 of the algorithm.
   - (Line 2.2.1) `op2 = X` (pop 1)
   - (Line 2.2.2) `op1 = res3` (pop 1)
   - (Line 2.2.3) `res4 = res3 + X` (res4)
   - (Line 2.2.4) push res4 in stack.
10. Scan `Y`, it's an operand. Line 2.1
11. Scan `A`, it's an operand. Line 2.1
12. Scan `B`, it's an operand. Line 2.1
(8) Scan `^`, it's an operator (line 2.9)

(line 2.9.1) op2 = B → B
(line 2.9.2) op1 = A → y
(line 2.9.3) res3 = A ∨ B → res

(line 2.9.4) push res3 in stack

(9) Scan `*`, it's an operator (line 2.9)

(line 2.9.1) op2 = res3 → res
(line 2.9.2) op1 = y → res
(line 2.9.3) res6 = y * res3 → res
(line 2.9.4) push res6 in stack

(10) Scan `–`, it's an operator (line 2.9)

(line 2.9.1) op2 = res6 → res
(line 2.9.2) op1 = res3 → res
(line 2.9.3) res7 = res3 - res6 → res
(line 2.9.4) push res7 in stack

(11) The expression is complete, pop result from stack (line 3

Result: res7 = res3 - res6

\[
\begin{align*}
\text{res7} &= \frac{(\text{res3 + X}) - (y \times \text{res5})}{\frac{\left(\text{res2} * E\right)}{\left(A \lor B\right)}} \\
&= \frac{\frac{\text{res1} \times C}{\left((A \land D) \land C\right)}}{(y \times (A \lor B))}
\end{align*}
\]

Example of values:

A = 4, D = 2, C = 1, E = 10

\[
\begin{align*}
\text{res7} &= (\left(\left(4 \land 2\right) \lor 1\right) * 10) - (1 * (4 \lor 1)) \\
&= 0 - 4
\end{align*}
\]