Expression is collection of operands and operators.
For example:

> x + y
> x is operand 1
> operation is +
> y is operand 2

Operators Precedence:
^
/ *
+ -
the others like in Java.

^ greater than / *
/ * greater than + -

To convert infix to post fix:
You put a table with 4 columns
1st: token number (token is either operand or operator)
2nd: token
3rd: operators stack
4th: post fix expression

if the token is operand you added directly to the post fix expression.
If the token is operator you check the top operator in the stack,

> if the new operator is greater than(>) the top operator in the stack, you push the new operator into the operator stack.
>
> if the new operator is less than or equal to (<=) the top operator in the stack you pop the top operator from the stack (keep doing pop until you reach operator in the stack > new operator).

Check HW3 Problem 1 no 1.

To convert post fix to infix:
You put a table with 4 columns
1st: token number (token is either operand or operator)
2nd: token
3rd: Stack (will hold the infix expression).

If the token is operand you added directly to the post fix expression.
If the token is operator you pop the top two operands from the stack and pit in between the operation, then push the new operand (do not calculate the operation, put it as is) for example if the stack contains 3 operands (2,5,4) and you have operator +, you pop 5 and 4 then you compose the new element (5 + 4), the new stack will be (2,(5+4))

if you have – or / or ^ the deeper element is the first operand.
For example if we have stack with (4,9,3) you want to push /, then you take 9 as first operand and 3 second operand, stack will be (4,(9/3)).

Check HW3 Problem 1 no 3.

Trace and evaluate post fix expression:
You put a table with 4 columns
1st: token number (token is either operand or operator)
2nd: token
3rd: Stack for operands only.

If the token s operand push it into the stack.
If the token is operator, pop the top two operand and calculate them with the operator, then push the reslut in the stack.

if you have – or / or ^ the deeper element is the first operand.
For example if we have stack with (4,9,3) you want to calculate /, then you take 9 as first operand and 3 second operand, stack will be (4,3).

Check HW3 Problem 1 no 2.

Trace and evaluate infix expression:

You put a table with 4 columns

1st: token number (token is either operand or operator)

2nd: token

3rd: operands stack

4th: operators expression

If the token is operand you added directly to the operand stack.

If the token is operator you check the top operator in the operator stack,

> if the new operator is greater than(>) the top operator in the operator stack, you push the new operator into the operator stack.
>
> if the new operator is less than or equal to (<=) the top operator in the operator stack you pop the top operator from the stack (keep doing pop until you reach operator in the stack > new operator) and the top two operands and calculate , push the result into the operands stack.

Check HW3 Problem 1 no 4.