1. Operator precedence parsing is a technique used to parse mathematical expressions or programming language statements based on the priority or precedence of operators.
2. Each operator is assigned a precedence level, which determines its priority in the expression. Operators with higher precedence are evaluated before operators with lower precedence.
3. The parsing process starts with the input expression and proceeds from left to right.
4. The parser maintains a stack to store operators and operands encountered during parsing.
5. As each token (such as numbers, operators, or parentheses) is read from the input expression, the parser compares its precedence with the top operator on the stack.
6. If the new token has higher precedence, it is pushed onto the stack.
7. If the new token has lower precedence, the stack is popped, and the corresponding operation is performed. This process continues until the stack is empty or a token with higher precedence is encountered.
8. The parsing process continues until all tokens in the input expression have been processed.
9. At the end of the parsing process, the stack contains the final result of the expression.
10. The operator precedence parsing technique is simple and efficient, especially for expressions with binary operators. It helps ensure that operators are evaluated in the correct order according to their precedence.
11. However, it may face challenges when dealing with unary operators or more complex language constructs. In such cases, additional parsing techniques, like operator associativity or grammar rules, may be used to handle these situations.

## Example:

A grammar G is said to be operator precedence if it possess following properties:

1. No production on the right side is Ɛ.
2. There should not be any production rule possessing 2 adjacent non terminals at the right hand side.

Consider the grammar for arithmetic expression

```
E→ EAE | (E) | -E | id
A → + | * | / | - | ^
```

This grammar is not an operator precedent grammar as in the production rule

```
E → EAE
```

It contains 2 consecutive non terminals.

Hence we first convert it into equivalent operator precedence grammar by removing A.

```
E → E+E | E*E | E/E | E-E | E^E
E → (E) | -E | id
```

In Operator precedence parsing we first define precedence relations <· = and ·> between pair of terminals.

The meaning of these relations is

1. p <· q means p gives more precedence than q.
2. p ·> q means p takes precedence over q.
3. p=q means has same precedence as q.

## Advantages of operator precedence parsing

1. This type of parsing is simple to implement.
2. Operator precedence parser is the only parser which can parse ambiguous grammar also.

## Disadvantages of operator precedence parsing

1. This type of parsing can be applicable for only small class of grammars.

## Application

The operator precedence parsing is done in a language having operators.

Hence in SNOBOL operator precedence parsing is done.