Operator Precedence
Operator Precedence Grammar:

- A grammar G is said to be operator precedence if it possess following two properties:
  - No production on the right side is E.
  - There should not be any production rule possessing two adjacent non-terminals at the right hand side.

- It is a small but important class of grammars.

- An operator precedence parser is a bottom-up parser that interprets an operator precedence grammar.
Example:

\[ E \rightarrow EAE / (E) / -E / id \]
\[ A \rightarrow + / - / * / 1 / ^{\wedge} \]

Not an operator precedence grammar

An operator precedence grammar

\[ E \rightarrow E + E / E - E / E * E / E / E / E ^{E} / (E) / -E / id \]
Designing Operator Precedence Parser:

In operator precedence parsing, we first define three disjoint precedence relations between every pair of terminals and construct the operator precedence table.

- $a < b$ if $b$ has higher precedence than $a$
- $a \equiv b$ if $b$ has same precedence as $a$
- $a > b$ if $b$ has lower precedence than $a$
Rules to determine precedence relations:

The determination of correct precedence relations between terminals are based on the traditional notions of associativity and precedence of operators.

Remember:

- id has higher precedence than any other symbol.
- $ has lowest precedence.
- If two operators have equal precedence, then we check the associativity of that particular operator.
Parsing the given string:

**Step-1:** Insert:
- $ symbol at the start and at the end of input string
- Precedence operator in between every two symbols of the string by referring the designed precedence table.

**Step-2:** Start scanning the string from left until seeing $ and put a pointer on its location. Now, scan backwards the string from right to left until seeing $. Everything between the two relations $ and $ forms the handle. Replace handle with the head of the respective production.

Repeat this step until reaching start symbol.
**Problem-01:**

Construct operator precedence parser for the following grammar:

\[ E \rightarrow E \cdot A \cdot E \mid id \]

\[ A \rightarrow + \mid / \mid * \]

Then, parse the following string:

\[ id + id \times id \]

www.gatevidyalay.com
Solution:

Step-1: Convert the given grammar to operator precedence grammar.

The equivalent operator precedence grammar is:

\[ E \rightarrow E + E \mid E \times E \mid \text{id} \]

Step-2: Construct the operator precedence table.

The terminal symbols we will consider are:

\{ id, +, *, $ \}
<table>
<thead>
<tr>
<th></th>
<th>id</th>
<th>+</th>
<th>*</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>+</td>
<td>&lt;</td>
<td>&gt;</td>
<td>&lt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>*</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>$</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
</tr>
</tbody>
</table>

Operator Precedence Table
Parsing the given string:

Step 1:

\[ id + id * id \]

(Given string to parse)

Insert $ at the two ends of the string

$ id + id * id $

Insert precedence operators in between the string

$<id> + <id> * <id>$
Step-2:

Scanning and Parsing:

$ <id> + <id> * <id> <$

$ E + <id> * <id> <$

$ E + E * <id> <$

$ E + E * E <$

$ + * $

$ <+ <*>_$

$ <+ > $
Problem-02:

Construct operator precedence parser for the grammar:

\[ S \rightarrow (L) / a \]
\[ L \rightarrow L, S / S \]

Also, parse the following string:

\( (a, (a, a)) \)

Solution:

The terminal symbols in the given grammar are:

\( \{ (, ), a, , \} \)

We will build the operator precedence table for these operators.
<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>(</th>
<th>)</th>
<th>,</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>(</td>
<td>&lt;</td>
<td>&gt;</td>
<td>:</td>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>)</td>
<td>&lt;</td>
<td>&gt;</td>
<td>:</td>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>,</td>
<td>&lt;</td>
<td>:</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>$</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
</tr>
</tbody>
</table>

Operator Precedence Table
Parsing the given string:

Step 1:

1. $(a, (a, a))$
2. Given string
3. Insert $\$\$ at the two ends of the string
   \[\$(a, (a, a))\$\]
4. Insert precedence operators in between the string
   \[\langle (\langle a \rangle, \langle a \rangle), \langle a \rangle \rangle \rangle \$\]
Step 2:

Scanning and Parsing:

$\langle (\langle a \rangle, \langle (\langle a \rangle, \langle a \rangle) \rangle) \rangle \rangle$

$\langle (s, \langle (\langle a \rangle, \langle a \rangle) \rangle) \rangle \rangle$

$\langle (s, \langle (s, \langle a \rangle) \rangle) \rangle \rangle$

$\langle (s, \langle (s, s) \rangle) \rangle \rangle$

$\langle (s, \langle (s) \rangle) \rangle \rangle$

$\langle (s, s) \rangle \rangle$
\[ < (L, s) > \]
\[ < (L) > \]
\[ s \]

---

Gate Vidyalay
A Temple of Learning
www.gatevidyalay.com
Advantages:

- This type of parsing is simple to implement.
- This parser is powerful enough for expressions in programming languages.

Disadvantages:

- It is hard to handle tokens like minus sign which has two different precedence depending on whether it is unary or binary.
- Only small class of grammars can be parsed using operator precedence parsing.
Operator Precedence Functions:

The operator precedence parsers usually do not store the precedence table with the relations. Rather, they are implemented in a special way.

Operator precedence parsers use precedence functions that map terminal symbols to integers and so the precedence relations between the symbols are implemented by numerical comparison.
Problem-03: Consider the following grammar:

\[ E \rightarrow E + E \mid E \times E \mid \text{id} \]

A) Construct operator precedence parser.
B) Find the operator precedence function.

Solution:
The terminal symbols we will consider for constructing the operator precedence table are:

\[ \{ +, \times, \text{id}, $\} \]
The graph representing the precedence function is:

<table>
<thead>
<tr>
<th>Operator</th>
<th>id</th>
<th>+</th>
<th>*</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>&lt;</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>+</td>
<td>&lt;</td>
<td>&gt;</td>
<td>&lt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>*</td>
<td>&lt;</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>$</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
</tr>
</tbody>
</table>

Operator Precedence Table
The longest possible paths are:

1. \( f_{id} \rightarrow g_{*} \rightarrow f_{+} \rightarrow g_{+} \rightarrow f_{\$} \)

2. \( g_{id} \rightarrow f_{*} \rightarrow g_{*} \rightarrow f_{+} \rightarrow g_{+} \rightarrow f_{\$} \)

The resulting precedence functions are:

<table>
<thead>
<tr>
<th></th>
<th>( + )</th>
<th>( * )</th>
<th>id</th>
<th>( $ )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f )</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>( g )</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>