Operator Precedence Parsing (Bottom Up parser)

It is used to define the mathematical operators for the compiler

Operator grammar :-

Example①

① $E \rightarrow E+E \mid E*E \mid id$

The above grammar is operator grammar because no two variables are adjacent

But

② $E \rightarrow EAE \mid id$
$A \rightarrow + \mid *$
$\Rightarrow$ $E \rightarrow E+E \mid E*E \mid id$
$A \rightarrow + \mid *$
⇒ This is operator grammar

~~This~~ ⇒ Both grammar ① and ② are same but the difference is that the two variables are adjacent in ② grammar and ∴ it is not operator grammar.

Example②

$S \rightarrow SAS \mid a$
$A \rightarrow bSb \mid b$

The above grammar is not operator grammar because two variables are adjacent

∴ convert to operator grammar

$S \rightarrow SbSbS \mid SbS \mid a$
$A \rightarrow bSb \mid b$

This is operator grammar

Operator precedence parser ~~never~~ can do the parsing by constructing operator ~~relation~~ precedence table.

for example

$E \to E+E \mid E*E \mid id$

	id	+	*	$
id	—	·>	·>	·>
+	<·	·>	<·	·>
*	<·	·>	·>	·>
$	<·	<·	<·	—

⇒ operator precedence table

① id > +

identifier will be given higher precedence ∴ (id > +)

② Two id's can never be compared because they cannot be adjacent ~~≡~~ ~~id~~.

③ id > *

(same reason as ①)

④ id > $

(same reason as ①)

⑤ + +

Since + is left associative ∴ Highest precedence will be give to left +.

∴ + > +

6 + <· *

7 + > $

$ will have least precedence when compared with any operator.

8 * > *

It is left associative ∴ left * will get highest precedence

9 $ $

completely successful.