OPERATOR PRECEDENCE PARSER

- A grammar G is said to be operator precedence if it posses following properties.
  (1) No production on the right side is ε
  (2) There should not be any production rule possessing two adjacent non-terminals at the right hand side.
- Consider the grammar for arithmetic expressions.

$E \rightarrow EAE \mid (E) \mid -E \mid id$
$A \rightarrow + \mid - \mid * \mid / \mid \wedge$

This grammar is not an operator precedence grammar as in the production rule.

$E \rightarrow EAE$

It contains two consecutive non-terminals. Hence first we will convert it into equivalent operator precedence grammar by removing A.

$E \rightarrow E+E \mid E-E \mid E*E \mid E/E \mid E \wedge E$
$E \rightarrow (E) \mid -E \mid id.$

- In OPP we will first define precedence relations $\lessdot$ $\doteq$ and $\gtrdot$ between pair of terminals. The meaning of these relation is

$p \lessdot q$	p gives more precedence than q.
$p \doteq q$	p has same precedence as q
$p \gtrdot q$	p takes precedence over q.

* Steps for operator Precedence Parsing OPP :-

1) Compute LEADING and TRAILING
2) Compute the precedence relations.
3) Fill the OPP table.
4) Insert the precedence relations in the i/p string.
5) Procss the i/p string using OPP table.
   (1) Scan the string from left until seeing $\gtrdot$
   (2) Scan the string backwards from right to left until seeing $\lessdot$
   (3) Everything between the two relations $\lessdot$ and $\gtrdot$ forms the handle
6) Compute the precedence function graph.
7) Find numerical value for f and g.

* Detailed steps for solving OPP

I) Computation of LEADING() and TRAILING()

LEADING(A)

1) a is in LEADING(A) if there is a production of the form $A \to \gamma a \delta$, where γ is either empty or a single nonterminal symbol.

2) If a is in LEADING(B) and there is a production of the form $A \to B\alpha$ then a is in leading of A.

TRAILING(A)

1) a is in TRAILING(A) if there is a production of the form $A \to \gamma a \delta$, where δ is either empty or single nonterminal symbol.

2) If a is in trailing(B) and there is a production of the form $A \to \alpha B$ then a is in trailing(A).

II) Computation of precedence relations.

i/p:- An operator grammar G

o/p:- relations $\lessdot$ $\doteq$ $\gtrdot$ for G

Method:-(1) Compute LEADING(A) and TRAILING(A) for each nonterminal A.

2) Execute by examining each position of the right side of each production.

3) set $\$ \lessdot a$ for all a in LEADING(S) [S-start symbol]
set $b \gtrdot \$$ for all b in TRAILING(S)
for each production $A \to X_1 X_2 \cdots X_n$ do
for i = 1 to n-1 do
begin
if X_i and X_{i+1} are both terminals then set $X_i \doteq X_{i+1}$;
X_i and X_{i+2} are terminals and X_{i+1} is a nonterminal then set $X_i \doteq X_{i+2}$
if X_i is a terminal and X_{i+1} is a nonterminal then for all a in LEADING(X_{i+1}) do set $X_i \lessdot a$;
if X_i is a nonterminal and X_{i+1} is a terminal then for all a in TRAILING(X_i) do set $a \gtrdot X_{i+1}$.

End

4) All the undefined entries are errors.

1) $ <· Leading(s)
   Trailing(s) ·> $
2) X_i X_{i+1} s.t. T_1 T_2 $\Rightarrow$ $T_1 \doteq T_2$ ie $X_i \doteq X_{i+1}$
3) X_i X_{i+1} X_{i+2} s.t. T_1 NT T_2 $\Rightarrow$ $T_1 \doteq T_2$
4) X_i X_{i+1} s.t. T NT $\Rightarrow$ T <· leading(NT)
5) X_i X_{i+1} s.t. NT T $\Rightarrow$ Trailing(NT) ·> T

III Precedence Table.

(*) The precedence relations existing among the terminals of any grammar G can be conveniently represented in a table called precedence table.

(*) Whether the precedence relations among the terminals of G are disjoint or not can be found by constructing the precedence table.

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

IV Operator precedence parsing Algorithm. (Pg 206 Aho Ullman)

I/P :- An input string ω and a table of precedence relations.

O/P :- If ω is well formed, a skeletal parse tree, with a placeholder nonterminal E labeling all interior nodes, otherwise an error indication.

Method :- Initially the stack contains $ and the I/P buffer the string ω$. To parse we execute the foll. program.

1) Set I/P to point to the first symbol of ω$;
2) Repeat forever
3) if $ is on top of the stack and I/P points to $ then
4) return
   else begin

5) Let a be the topmost terminal symbol on the stack and let b be the symbol pointed to by input;
6) if $a \lessdot b$ or $a \doteq b$ then begin
7) Push b onto the stack;
8) Advance i/p to the next input symbol;
end;
9) else if $a \gtrdot b$ then
10) repeat
11) Pop stack
12) until the top stack terminal is related by $\lessdot$ to the terminal most recently popped.
13) else error ()
End.

IV) Algorithm for constructing precedence functions graph.

Precedence Functions.

- Compilers using operator precedence parsers need not store the table of precedence relations.
- In most cases the table can be encoded by 2 precedence functions f and g which maps terminal symbols into integers
- we attempt to select f and g for terminal symbols a and b.
  1) $f(a) < g(b)$ whenever $a \lessdot b$
  2) $f(a) = g(b)$ whenever $a \doteq b$ and
  3) $f(a) > g(b)$ whenever $a \gtrdot b$.
- Thus the precedence relation between a and b can be determined by a numerical comparison between $f(a)$ and $g(b)$

Method for finding Precedence functions for a table :—

1) Create symbols f_a and g_a for each a ie a terminal or $.
2) Partition the created symbols into as many groups as possible in such a way that if $a \doteq b$ then f_a and g_b are in the same group.

$a \doteq b$
f_a, g_b
same group

$a \lessdot b$ $\quad g_b \Rightarrow f_a$.

$a \gtrdot b$ $\quad f_a \Rightarrow g_b$

- We may have to put symbols in the same group even if they are not related by $\doteq$
eg:- if $a \doteq b$ and $c \doteq b$ then f_a and f_c must be in the same group as g_b.

3). Create a directed graph whose nodes are the groups found in step 2. For any a and b if $a \lessdot b$, place an edge from the group of g_b to the group of f_a. If $a \gtrdot b$ place an edge from the group of f_a to that of g_b.

$a \lessdot b$ $g_b \to f_a$
$a \gtrdot b$ $f_a \to g_b$

4) If the graph constructed in step 3 has a cycle, then no precedence functions exist. If there are no cycles, let $f(a)$ be the length of the longest path beginning at the group of f_a; let $g(a)$ be the length of the longest path from the group of g_a. ie collect the length of the longest paths from the groups of f_a and g_b respt.

f_a, g_b collect the length of the longest paths from f_a & g_b.

Examples (Problem)

(*) Consider the grammar G

$E \to E + T$

$E \to T$

$T \to T * F$

$T \to F$

$F \to (E)$

$F \to id.$

Find OPP parser.

Solution → find leading and Trailing of non-terminals.

The terminals are = { +, *, id, (,) }.

The non-terminals are = { E, T, F }.

leading (E) = { +, *, (, id }

leading (T) = { *, (, id }

leading (F) = { (, id }.

Trailing (E) = { *, +,), id }

Trailing (T) = { *,), id }

Trailing (F) = {), id }

- Next let us compute the precedence relations.

Here E is the start symbol.

$\$ \lessdot a$ for all a in LEADING(E)

leading E = { +, *, (, id }.

$\$ \lessdot +$

$\$ \lessdot *$

$\$ \lessdot ($

$\$ \lessdot id.$

Similarly, set $b \gtrdot \$$ for all b in Trailing(E).

Trailing(E) = { *, +,), id }

$* \gtrdot \$$

$+ \gtrdot \$$

$) \gtrdot \$$

$id \gtrdot \$$

Take $E \to E + T$

$E \to E + T$ (NT, T) $\overset{set}{\Rightarrow}$ Trailing (NT) $\gtrdot$ T

Trailing (E) $\gtrdot$ +

ie: $* \gtrdot +$

$+ \gtrdot +$

$) \gtrdot +$

$id \gtrdot +$

$E \to E + T$ (T, NT) $\overset{set}{\Rightarrow}$ T $\lessdot$ leading (NT)

+ $\lessdot$ leading (T)

$+ \lessdot *$

$+ \lessdot ($

$+ \lessdot id.$

Take $T \to T * F$

$T \to \underset{NT}{T} * \underset{T}{F}$ सूत्र Trailing(NT) $\doteq$ T

Trailing(T) $\doteq$ *

$* \doteq *$

$) \doteq *$

$id \doteq *$

$T \to T * F$ सूत्र $\Rightarrow$ T $\lessdot$ leading(NT)

$\underset{T}{*} \lessdot$ leading(F)

$* \lessdot$ (

$* \lessdot$ id

$F \to (E)$ सूत्र $\Rightarrow$ T $\lessdot$ leading(NT)

$\underset{T}{(} \lessdot$ leading(E)

$(\lessdot +$

$(\lessdot *$

$(\lessdot ($

$(\lessdot id$

$F \to (E)$ सूत्र Trailing(NT) $\doteq$ T

$\underset{NT}{E} \underset{T}{)}$ Trailing(E) $\doteq$)

$* \doteq)$

$+ \doteq)$

$) \doteq)$

$id \doteq)$

$F \to (E)$ सूत्र $T_1 \doteq T_2$

T NT T ie ($\doteq$)

Construct the precedence table.

	+	*	(	)	id	$
+	$\doteq$	$\lessdot$	$\lessdot$	$\doteq$	$\lessdot$	$\doteq$
*	$\doteq$	$\doteq$	$\lessdot$	$\doteq$	$\lessdot$	$\doteq$
(	$\lessdot$	$\lessdot$	$\lessdot$	$\doteq$	$\lessdot$	
)	$\doteq$	$\doteq$	>	$\doteq$	$\lessdot$	$\doteq$
id	$\doteq$	$\doteq$		$\doteq$		$\doteq$
$	$\lessdot$	$\lessdot$	$\lessdot$		$\lessdot$	

All the undefined entries are errors.

$\Rightarrow$ Take an input string from the grammar & process it.

Input string id + id * id.

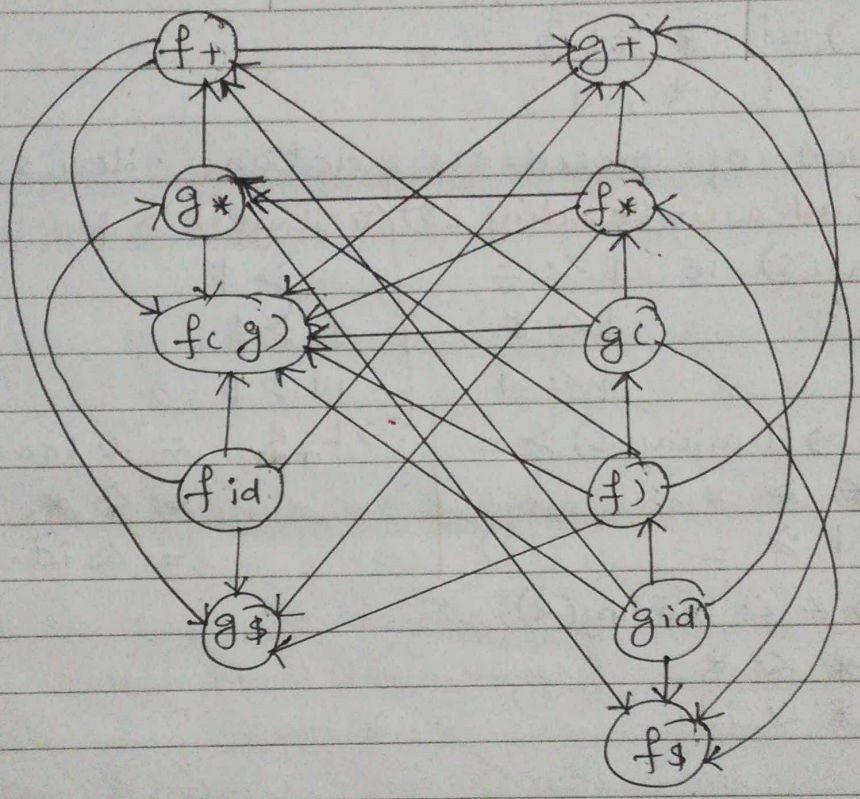
Stack	Input string	Action
$	id + id * id $	$\lessdot$ $\Rightarrow$ shift
$ $\lessdot$ id	+ id * id $	$\doteq$ $\Rightarrow$ Reduce; pop
$	+ id * id $	$\lessdot$ $\Rightarrow$ shift
$ $\lessdot$ +	id * id $	$\lessdot$ $\Rightarrow$ shift
$ $\lessdot$ + $\lessdot$ id	* id $	$\doteq$ $\Rightarrow$ Reduce; pop
$ $\lessdot$ +	* id $	$\lessdot$ $\Rightarrow$ shift
$ $\lessdot$ + $\lessdot$ *	id $	$\lessdot$ $\Rightarrow$ shift

Stack	Input string	Action
$ <· + <· * <· id	>$	> ⇒ Reduce, pop.
$ <· + <· *	>$	> ⇒ Reduce, pop.
$ <· +	>$	> ⇒ Reduce, pop.
$	$	ACCEPT

⇒ Draw the precedence graph.

a <· b
g ⇒ f.

a ·> b
f ⇒ g.



	+	*	id	(	)	$
f	2	4	4	0	4	0
g	1	3	5	5	0	0