Assignment No.9

AIM:

Implement Operator Precedence Parser

THEORY:

The grammars which have the property that no production right side is epsilon or has two adjacent non-terminals is called an operator grammar.

The following grammar for expressions

$E \rightarrow EAE \mid (E) \mid -E \mid id$

$A \rightarrow + \mid - \mid * \mid / \mid \uparrow$

is not an operator grammar, because the right side EAE has two (in fact three) consecutive nonterminal, However, if we substitute for A each of its alternatives, we obtain the following operator grammar;

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

This parser relies on the following three precedence relations: **⋖, ≐, ⋗**

a ⋖ b This means a "yields precedence to" b.

a ⋗ b This means a "takes precedence over" b.

a ≐ b This means a "has precedence as" b.

	id	+	*	$
id		⋗	⋗	⋗
+	⋖	⋗	⋖	⋗
*	⋖	⋗	⋗	⋗
$	⋖	⋖	⋖	

Figure 1.Operator precedence relation table for grammar E->E+E/E*E/id

There is not given any relation between id and id as id will not be compared and two variables cannot come side by side. There is also a disadvantage of this table as if we have n operators than size of table will be n*n and complexity will be $0(n^2)$. In order to increase the size of table, use **operator function table**.

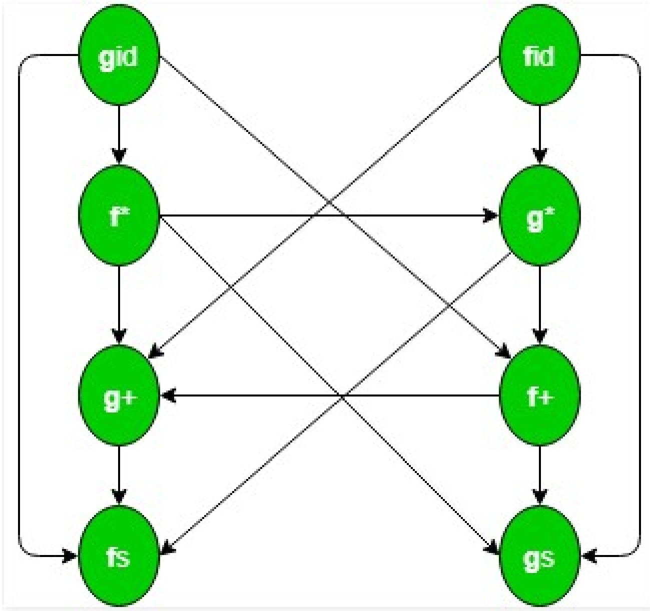
The operator precedence parsers usually do not store the precedence table with the relations; rather they are implemented in a special way. Operator precedence parsers use **precedence functions** that map terminal symbols to integers, and so the precedence relations between the symbols are implemented by numerical comparison. The parsing table can be encoded by two precedence functions **f** and **g** that map terminal symbols to integers. We select f and g such that:

1. f(a) < g(b) whenever a is precedence to b
2. f(a) = g(b) whenever a and b having precedence
3. f(a) > g(b) whenever a takes precedence over b

Example – Consider the following grammar:

E -> E + E/E * E/(E)/id

The directed graph representing the precedence function:



Since there is not any cycle in the graph so we can make function table:

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

PROGRAM:

INPUT AND OUTPUT:

Operator precedence parser

====================

Enter the string(Use i for an identifier)i*(i+i)-i/(i^i)

$ i*(i+i)-i/(i^i)$

$i *(i+i)-i/(i^i)$

$E *(i+i)-i/(i^i)$

$E*	(i+i)-i/(i^i)$
$E*(	i+i)-i/(i^i)$
$E*(i	+i)-i/(i^i)$
$E*(E	+i)-i/(i^i)$
$E*(E+	i)-i/(i^i)$
$E*(E+i	)-i/(i^i)$
$E*(E+E	)-i/(i^i)$
$E*(E	)-i/(i^i)$
$E*E	-i/(i^i)$
$E	-i/(i^i)$
$E-	i/(i^i)$
$E-i	/(i^i)$
$E-E	/(i^i)$
$E-E/	(i^i)$
$E-E/(	i^i)$
$E-E/(i	^i)$
$E-E/(E	^i)$
$E-E/(E^	i)$
$E-E/(E^i	)$
$E-E/(E^E	)$
$E-E/(E	)$
$E-E/E	$
$E-E	$
$E	$

Accepted

CONCLUSION:

Operator Precedence Parser which have the property that no production right side is epsilon or has two adjacent non-terminals is implemented.

REFERENCES:

- Compilers - Principles, Techniques and Tools - A.V. Aho, R. Shethi and J. D. Ullman (Pearson Education)
- https://www.geeksforgeeks.org/theory-computation-operator-grammar-precedence-parser/