Experiment No.6

Title: - Operator Precedence Parser

Aim:- Implement the Bottom-Up *Operator Precedence Parser* for the given grammar.

Theory:-

1. Precedence Relations

Bottom-up parsers for a large class of context-free grammars can be easily developed using *operator grammars*.

Operator grammars have the property that no production right side is empty or has two adjacent nonterminals. This property enables the implementation of efficient *operator-precedence parsers*. These parser rely on the following three precedence relations:

Relation	Meaning
$a \lessdot b$	*a* yields precedence to *b*
$a \doteq b$	*a* has the same precedence as *b*
$a \gtrdot b$	*a* takes precedence over *b*

These operator precedence relations allow to delimit the handles in the right sentential forms: $\lessdot$ marks the left end, $\doteq$ appears in the interior of the handle, and $\gtrdot$ marks the right end.

Let assume that between the symbols a_i and a_{i+1} there is exactly one precedence relation. Suppose that \$ is the end of the string. Then for all terminals we can write: \$ $\lessdot$ *b* and *b* $\gtrdot$ \$. If we remove all nonterminals and place the correct precedence relation: $\lessdot$, $\doteq$, $\gtrdot$ between the remaining terminals, there remain strings that can be analyzed by easily developed parser.

For example, the following operator precedence relations canbe introduced for simple expressions:

	id	+	*	$
id		$\gtrdot$	$\gtrdot$	$\gtrdot$
+	$\lessdot$	$\gtrdot$	$\lessdot$	$\gtrdot$
*	$\lessdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$
$	$\lessdot$	$\lessdot$	$\lessdot$	$\gtrdot$

Example: The input string: $id_1 + id_2 * id_3$ after inserting precedence relations becomes

\$ $\lessdot$ id_1 $\gtrdot$ + $\lessdot$ id_2 $\gtrdot$ * $\lessdot$ id_3 $\gtrdot$ \$

Having precedence relations allows to identify handles as follows:

- scan the string from left until seeing ·>

- scan backwards the string from right to left until seeing <·

- everything between the two relations <· and ·> forms the handle

Note that not the entire sentential form is scanned to find the handle.

2. Operator Precedence Parsing Algorithm

Initialize: Set *ip* to point to the first symbol of *w*$

Repeat: Let *X* be the top stack symbol, and *a* the symbol pointed to by *ip*

if $ is on the top of the stack and ip points to $ then return

else

Let *a* be the top terminal on the stack, and *b* the symbol pointed to by *ip*

if *a* <· *b* or *a* =· *b* then

push *b* onto the stack

advance *ip* to the next input symbol

else if *a* ·> *b* then

repeat

pop the stack

until the top stack terminal is related by <·

to the terminal most recently popped

else *error()*

end

3. Making Operator Precedence Relations

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. Not every table of precedence relations has precedence functions but in practice for most grammars such functions can be designed.

Input Specifications:- Students are supposed to take input the operator precedence table in any of the languages C, C++, Java or Python.

Output Specifications:- Print the contents of stack, input buffer and action.

Exercise:-

Q1. In operator precedence parsing, precedence relations are defined

A. For all pair of non terminals

B. For all pair of terminals

C. To delimit the handle

D.Only for a certain pair of terminals

Answer :

Q2. For a given grammar below, construct an operator precedence relation matrix, assuming * and + are binary operatora and id as terminal symbol and E as non-terminal.

E -> E+E E-> E*E E-> id

Apply operator precedence parsing algorithm for the input string id+id*id.

Conclusion: