Q3) Write a short note on operator precedence parser.

Ans →An operator precedence parser is a type of ~~shift-reduce~~ parser used in compiler design for syntax analysis. It is based on the precedence of ~~parse~~ operators in the input language. The parser uses a stack to keep track of operators and operands, and a set of parsing rules to construct an abstract syntax tree from the input string.

→ The parsing algorithm works by reading the input string from left to right and pushing operators and operands onto the stack based on their precedence. When a lower-precedence operator is encountered, the parser reduces the stack by applying the appropriate parsing rule.

→ The main advantage of operator precedence parser is its simplicity and efficiency. It can handle a wide range of programming languages, including those with complex operator precedence rules. However, it has some limitations, such as the inability to handle left-recursive grammars and some forms of ambiguity in the input language.

→ Operator precedence parser is an important technique in compiler design that allows efficient and accurate syntax analysis of programming languages.

Advantages

1. Fast
2. Easy to understand and implement
3. Can handle a wide range of grammars
4. Can be used to parse complex expressions and equations

Disadvantages

① It cannot handle left recursion

② Not suitable for parsing languages with a large number of rules,

③ It requires more memory than other parsing techniques.

Example

E → EAE | id

A → + | x

construct operator precedence parser and parse the string id + id x id

Ans Converting the above grammar into operator precedence grammar

E → E+E | E*E | id

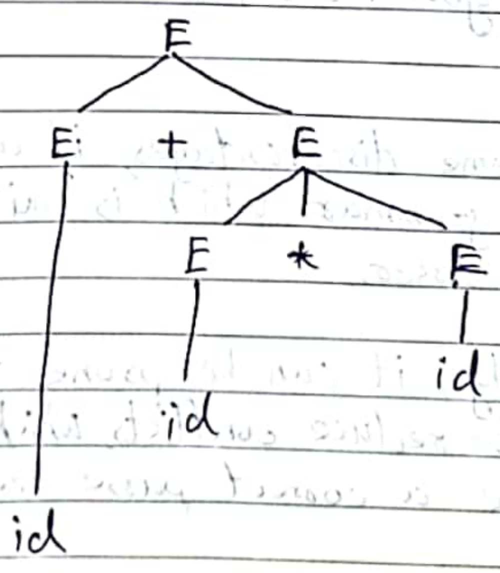
step① :- operator precedence table

	id	+	*	$
id	–	>	>	>
+	<	>	<	>
*	<	>	>	>
$	<	<	<	A

Step ②:- Operator precedence table

Stack	input	Comment
$	id + id *id $	Shift
$id	+id *id $	Reduce E→id
$E	+id *id $	Shift
$E+	id*id $	Shift
$E+ id	*id$	Reduce E→id
$E+E	*id$	Shift
$E+E*	id $	Shift
$E+E*id	$	Reduce E→id
$E+E*E	$	Reduce E→E*E
$E+E	$	Reduce E→E+E
$ E	$	Accept

Parse tree



Q4) Write short note on shift reduce parser.

Ans: → Shift reduce parser is a bottom up parser, it uses bottom up parsing for constructing the parse tree of an input string.

→ The parser initially shifts the input symbols onto a stack, then reduces the stack by applying production rules until start symbol is reached.

→ Shift reduce parser uses shift operation to move input symbols on the top of stack with a non-terminal symbol. The parser determines whether to shift or reduce based on the next input symbol and top of stack.

→ It is capable of parsing left-recursive and ambigous grammar, which others parsers may struggle with.

→ It has some disadvantages, it can be difficult to construct grammar which is suitable for shift-reduce parser.

→ Additionally, it can be prone to reduce-reduce and shift-reduce conflicts, which must be resolved to produce a correct parse tree

Example

Consider the grammar

S → (L) | a
L → L, S | S

Perform shift reduce parsing for input string "(a,(a,a))

Stack	~~Action~~ Input	Action
$	(a,(a,a))$	Shift
$(	a,(a,a))$	Shift
$(a	,(a,a))$	Reduce S→a
$(S	,(a,a))$	Reduce L→S
$(L	,(a,a))$	Shift
$(L,	(a,a))$	Shift
$(L,(	a,a))$	Shift
$(L,(a	,a))$	Reduce S→a
$(L,(S	,a))$	Reduce L→S
$(L,(L	,a))$	Shift
$(L,(L,	a))$	Shift
$(L,(L,a	))$	Reduce S→a
$(L,(L,S	))$	Reduce L→L,S
$(L,(L	))$	Shift
$(L,(L)	)$	Reduce S→(L)
$(L,S	)$	Reduce L→L,S
$(L	)$	Shift
$(L)	$	Reduce S→(L)
$S	$	Accept