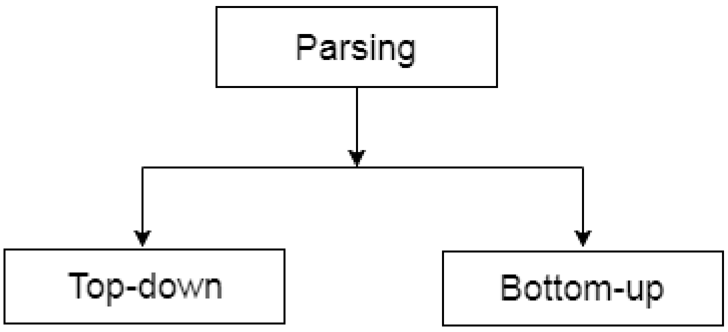
Parser

Parser is a compiler that is used to break the data into smaller elements coming from lexical analysis phase.

A parser takes input in the form of sequence of tokens and produces output in the form of parse tree.

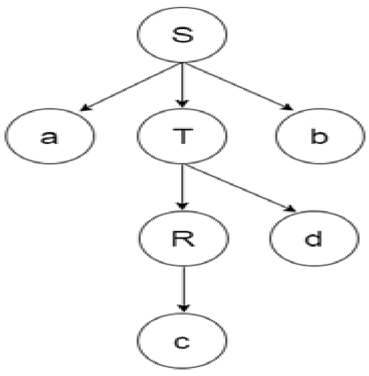
Parsing is of two types: top down parsing and bottom up parsing.



Top down paring

- The top down parsing is known as recursive parsing or predictive parsing.
- Bottom up parsing is used to construct a parse tree for an input string.
- In the top down parsing, the parsing starts from the start symbol and transform it into the input symbol.

Parse Tree representation of input string "acdb" is as follows:



Bottom up parsing

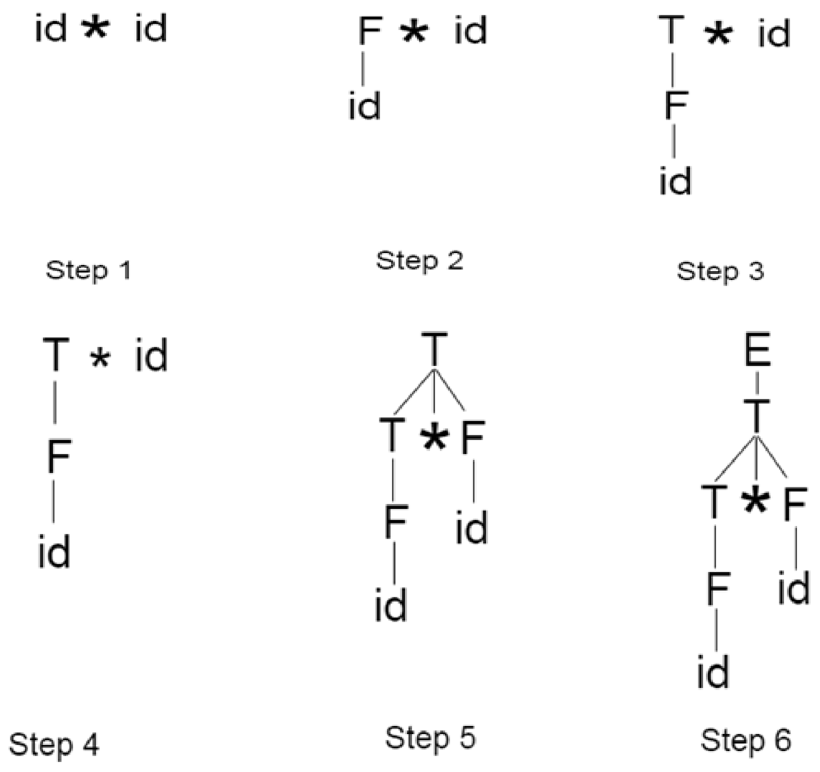
- Bottom up parsing is also known as shift-reduce parsing.
- Bottom up parsing is used to construct a parse tree for an input string.
- In the bottom up parsing, the parsing starts with the input symbol and construct the parse tree up to the start symbol by tracing out the rightmost derivations of string in reverse.

Example

Production

1. E → T
2. T → T * F
3. T → id
4. F → T
5. F → id

Parse Tree representation of input string "id * id" is as follows:



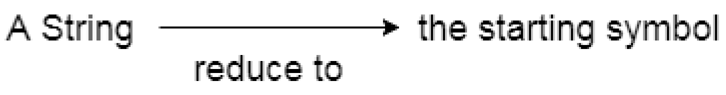
Bottom up parsing is classified in to various parsing. These are as follows:

1. Shift-Reduce Parsing
2. Operator Precedence Parsing
3. Table Driven LR Parsing

a. LR(1)
b. SLR(1)
c. CLR (1)
d. LALR(1)

Shift reduce parsing

- Shift reduce parsing is a process of reducing a string to the start symbol of a grammar.
- Shift reduce parsing uses a stack to hold the grammar and an input tape to hold the string.



- Sift reduce parsing performs the two actions: shift and reduce. That's why it is known as shift reduces parsing.
- At the shift action, the current symbol in the input string is pushed to a stack.
- At each reduction, the symbols will replaced by the non-terminals. The symbol is the right side of the production and non-terminal is the left side of the production.

Example:

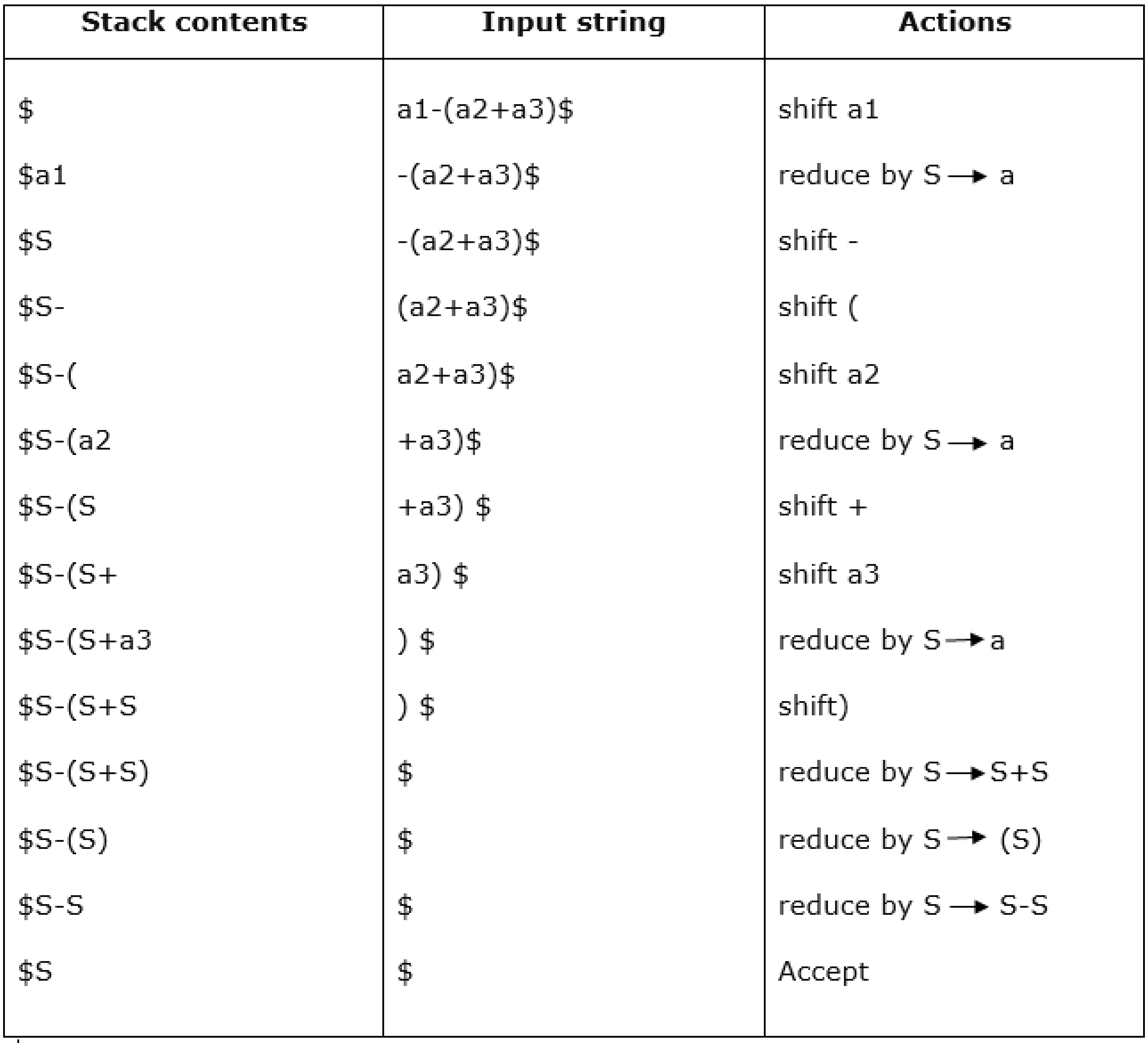
Grammar:

1. S → S+S
2. S → S-S
3. S → (S)
4. S → a

Input string:

1. a1-(a2+a3)

Parsing table:



Stack contents	Input string	Actions
\$	a1-(a2+a3)\$	shift a1
\$a1	-(a2+a3)\$	reduce by $S \rightarrow a$
\$S	-(a2+a3)\$	shift -
\$S-	(a2+a3)\$	shift (
\$S-(	a2+a3)\$	shift a2
\$S-(a2	+a3)\$	reduce by $S \rightarrow a$
\$S-(S	+a3) \$	shift +
\$S-(S+	a3) \$	shift a3
\$S-(S+a3	) \$	reduce by $S \rightarrow a$
\$S-(S+S	) \$	shift)
\$S-(S+S)	\$	reduce by $S \rightarrow S+S$
\$S-(S)	\$	reduce by $S \rightarrow (S)$
\$S-S	\$	reduce by $S \rightarrow S-S$
\$S	\$	Accept

There are two main categories of shift reduce parsing as follows:

1. Operator-Precedence Parsing
2. LR-Parser

Operator precedence parsing

Operator precedence grammar is kinds of shift reduce parsing method. It is applied to a small class of operator grammars.

A grammar is said to be operator precedence grammar if it has two properties:

- No R.H.S. of any production has a∈.
- No two non-terminals are adjacent.

Operator precedence can only established between the terminals of the grammar. It ignores the non-terminal.

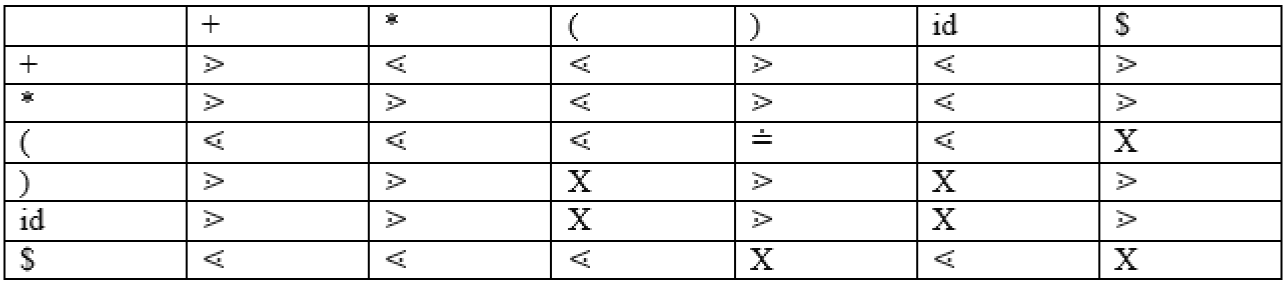
There are the three operator precedence relations:

a ⋗ b means that terminal "a" has the higher precedence than terminal "b".

a ⋖ b means that terminal "a" has the lower precedence than terminal "b".

a ≐ b means that the terminal "a" and "b" both have same precedence.

Precedence table:



	+	*	(	)	id	$
+	⋗	⋖	⋖	⋗	⋖	⋗
*	⋗	⋗	⋖	⋗	⋖	⋗
(	⋖	⋖	⋖	≐	⋖	X
)	⋗	⋗	X	⋗	X	⋗
id	⋗	⋗	X	⋗	X	⋗
$	⋖	⋖	⋖	X	⋖	X

Parsing Action

- Both end of the given input string, add the $ symbol.
- Now scan the input string from left right until the ⋗ is encountered.
- Scan towards left over all the equal precedence until the first left most ⋖ is encountered.
- Everything between left most ⋖ and right most ⋗ is a handle.
- $ on $ means parsing is successful.

Example

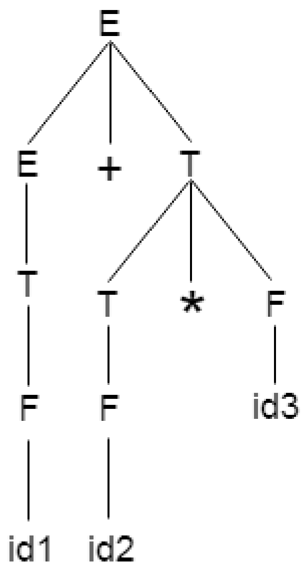
Grammar:

1. E → E+T/T
2. T → T*F/F
3. F → id

Given string:

1. w = id + id * id

Let us consider a parse tree for it as follows:



On the basis of above tree, we can design following operator precedence table:

	E	T	F	id	+	*	$
E	X	X	X	X	≐	X	⋗
T	X	X	X	X	⋗	≐	⋗
F	X	X	X	X	⋗	⋗	⋗
id	X	X	X	X	⋗	⋗	⋗
+	X	≐	⋖	⋖	X	X	X
*	X	X	≐	⋖	X	X	X
$	⋖	⋖	⋖	⋖	X	X	X

Now let us process the string with the help of the above precedence table:

$\$ \lessdot \mathrm{id1} \gtrdot + \mathrm{id2} * \mathrm{id3}\ \$$

$\$ \lessdot F \gtrdot + \mathrm{id2} * \mathrm{id3}\ \$$

$\$ \lessdot T \gtrdot + \mathrm{id2} * \mathrm{id3}\ \$$

$\$ \lessdot E \doteq + \lessdot \mathrm{id2} \gtrdot * \mathrm{id3}\ \$$

$\$ \lessdot E \doteq + \lessdot F \gtrdot * \mathrm{id3}\ \$$

$\$ \lessdot E \doteq + \lessdot T \doteq * \lessdot \mathrm{id3} \gtrdot \$$

$\$ \lessdot E \doteq + \lessdot T \doteq * \doteq F \gtrdot \$$

$\$ \lessdot E \doteq + \doteq T \gtrdot \$$

$\$ \lessdot E \doteq + \doteq T \gtrdot \$$

$\$ \lessdot E \gtrdot \$$

Accept.