Language Processing System:

Source Program
↓
Preprocessor
↓ Modified source program
Compiler
↓ target Assembly code
Assembler
↓ Relocatable machine code
Library files → Linker / Loader
Relocatable objects
↓
Target Machine code.

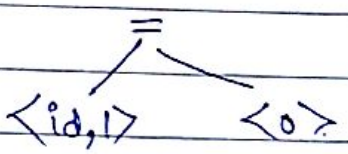
Lexical Analysis: use DFA

```
int a = 0;
int b = 1;
int c = 2;
int Sum = 0;
Sum = a+b+c;
```

lexime.

Role of Lexical Analysis:
1. Simply, we can remove white space, comment
2. with the help of DFA we can easily recognize token
3. complexity reduce.

S.No	Name	type	data type
1	a	id	int
2	b	id	int
3	c	id	int
4	Sum	id	int

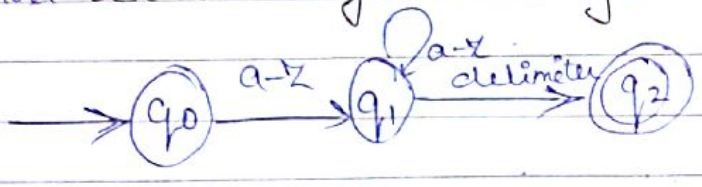


<id,1> <=> <0>
<id,2> <=> <1>
<id,3> <=> <2>

Syntax Analysis → use PDA (Push down Automata)
Semantic Analysis – Parsing

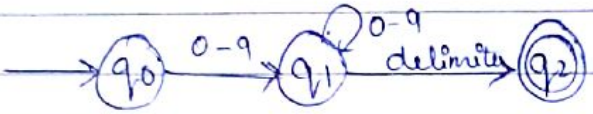
Page No.
Date : / /201

~~2x~~ Role Lexical Analysis :- using DFA.

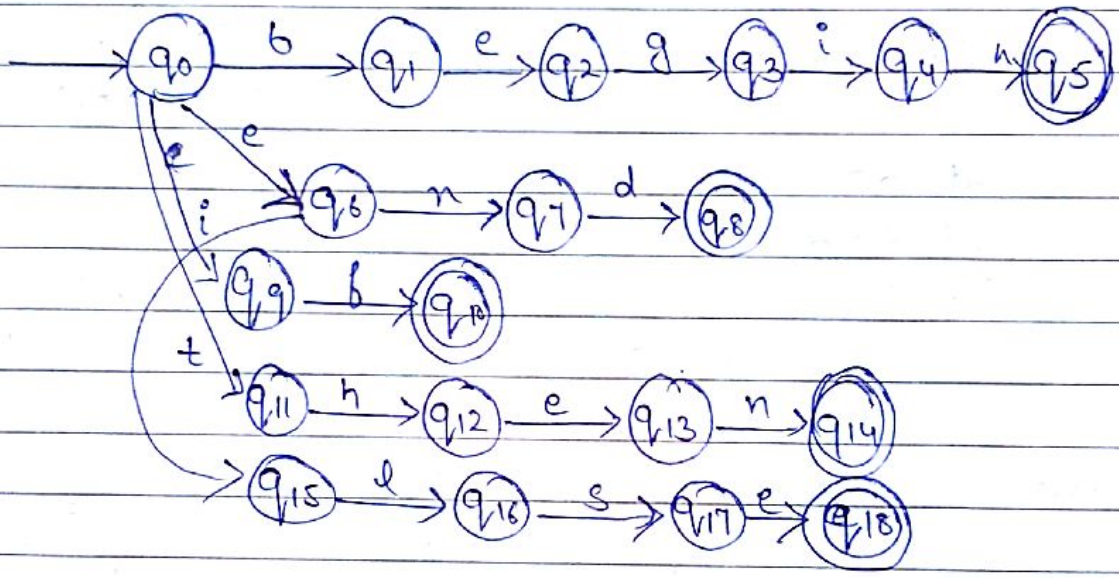


Regular Exp :-
$(a-z)(a-z+0-9)^*(del)$

For integer values.

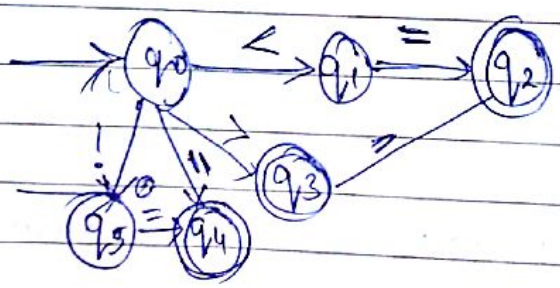


Keywords :- begin, end, if, then, else



Relational operator (relop).

$i < 2i$



Evergreen
Page No.
Date / /201

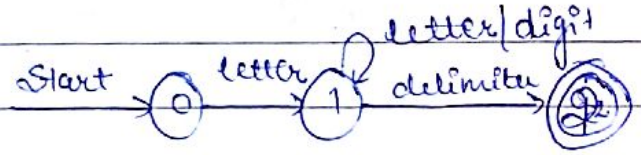
language → Regular Expression :-

Token Code Value
begin

Token	Code	Value
begin	1	—
end	2	—
if	3	—
then	4	—
else	5	—
identifier	6	Pointer to Symbol table

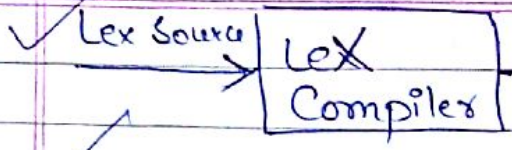
Token	Code	Value
Constant	7	Pointer to Symbol table
<	8	1
<=	8	2
=	8	3
<>	8	4
>	8	5
>=	8	6

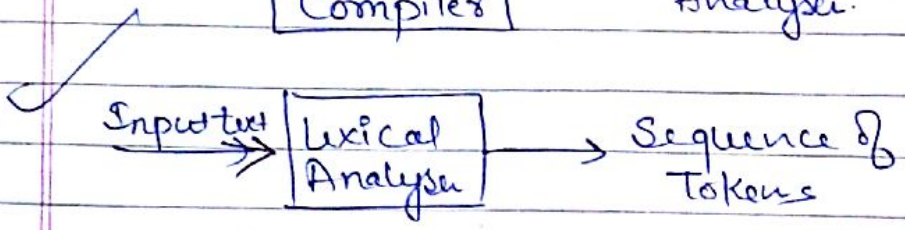
- Complexity decrease
- Acc to Presidence number is assign.

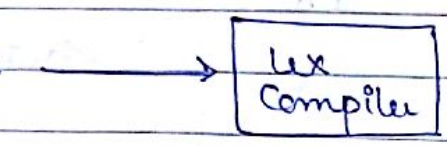
→ 

State 0: C = GETCHAR()
if LETTER(C) then go to state 1
else FAIL()

State 1: C = GETCHAR()
if LETTER(C) or DIGIT(C) then go to State 1
else if DELIMITER(C) go to State 2
else FAIL()







Format of Lex Auxillary :

$D_1 = R_1$

$D_2 = R_2$

Delimiter = { , - ; = + - }

If (BEGIN)
Return 1
If (END)
Return 2

Aux Def
letter = a-z
digit = 0-9
identifier = (letter/digit)*
integer = (digit)+
Sign = + | - | ∈
Signed integer = Sign integer

Lexical Analysis Implementation :

- DFA is used
- Regular Expression as a language is used
- Symbol table

Page No.
Date / /201

Syntax Analysis :-

1. CFG
2. Parse Tree
3. Parsing Techniques.

Type 0	Phrase Structure	Turing machine.
Type 1	Context Sensitive	—
Type 2	Context free	PDA
Type 3	Regular Grammar	DFA

$G = (N, T, P, S)$

N → Statement, Expression.
S E

T → operator (+, −, ÷, *,), (, ; a, b, c, id)

if statement { if(exp)-then state else state)

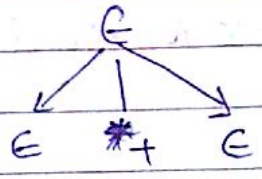
SL → S ; SL/S Statement list is collection of statment Sperated by semicolon.

Statement list → Statement ; Statement List / → Statement

Expression → Expression + Expression
E → E + E
E → 2 + 3

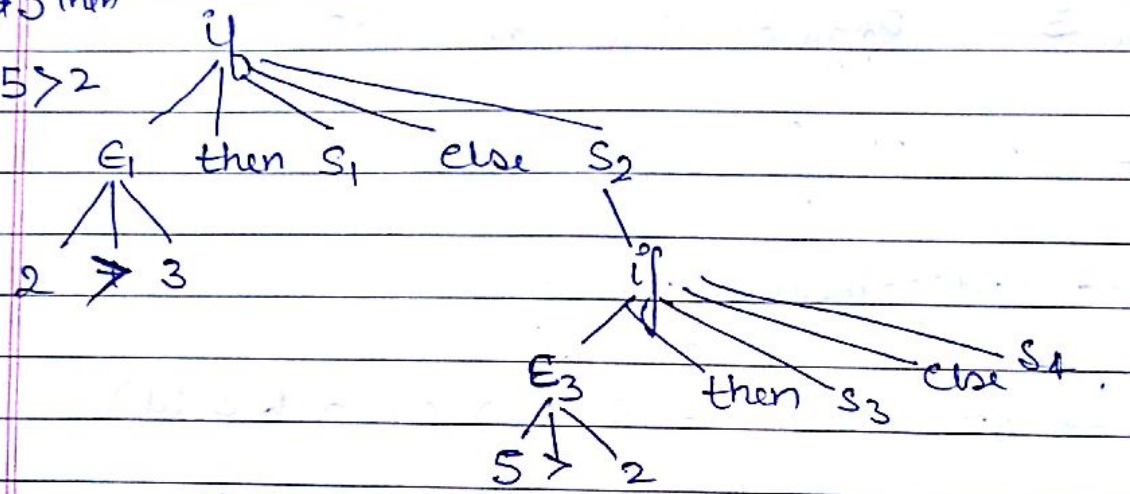
E → E+E | E*E | id | -E | (E)
→ E*E+E
→ id* E+E
→ id* id +E
→ id* id +id.

$E \rightarrow E * E$
$\rightarrow E + E * E$
$\rightarrow id + E * E$
$\rightarrow id + id * E$
$\rightarrow id + id * id.$



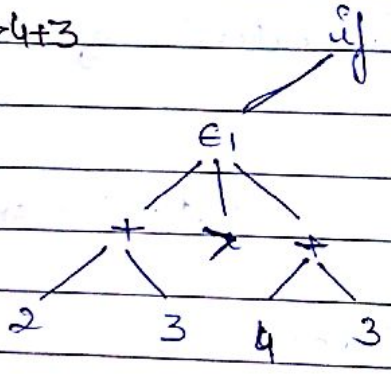
if E_1 then S_1 else S_2
↓
if E_3 then S_3 else S_4.

if $E_1 \rightarrow 2 > 3$ then
$E_3 \rightarrow 5 > 2$



Derivation tree.

if $E_1 \rightarrow 2+3 > 4+3$



Evergreen
Page No.
Date : / /201

Notational Conventions :-

1) Non Terminals :- Lower case name Such as expression, Statement and operator

2) Start Symbol (S) :- Non terminal

3) E ; S — Expression & Statement (grammer Symbol) → Non-Terminal Single letter in Capital

4) Terminal Symbol :- Single lower case letter a, b, c

5) Operator Symbol :- +, −, ÷, *,

6) Puncuation Symbol :- Such as paranthesis, digit 0-9

7) Identifier (id) :-

$\alpha\beta\gamma$ → Strings of Grammer Symbol. $(NUT)^*$

Lexical analysis → Regular Expression
Syntax analysis → Token as a input
Linker
Scanning → Lexical Analysis
Lexical Analysis Complier → output Lexical Analyser

Lexeme is a sequence of characters in the Source program that is matched by the pattern for a token.

Evergreen
Page No.
Date: / /201

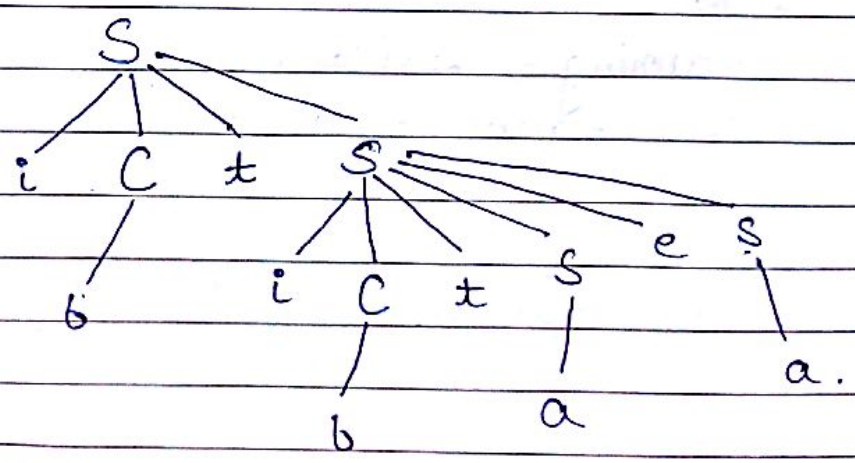
Parser :- A parser for gramner G is a program that takes as input the string w and produces as output either a parse tree for w, if w is a sentence of G or an error msg indicating that w is not a sentence of G

Two basic type of Parser
1) Bottom up parser
2) Top down parser

Representation of parse tree :-
(1) Implicit (Production rule)
(2) Explicit (parse tree)

Two type of derivations :-
(1) Left most derivation (2) Right most derivation or Canonical or Left Sentential form

$S \rightarrow iCtS$
$S \rightarrow iCtSeS$
$S \rightarrow a$
$C \rightarrow b$



w = ibtibtaea
↳ Left Sentential form

Bottom up:

Shift reduce parsing :- Shift reduce parsing is a bottom up parsing technique it attempts to construct a Parse tree for a input string begining at the leave And working up toward the root where root is a Start Symbol S.

At each step a string matching the right side of production is replaced by the symbol on left.

- Each replacement of the right side of a production by the left side in the process above is called the reduction
- Replacement of the substring by left side of production leads to the reduction to the Start Symbol.

Left :-

$w = ibtibtaeaa$
$= ictibtaeaa$
$= ictictaeaa$
$= ictictseaa$
$= ictictsesS$
$= ictictsS$
$= S$

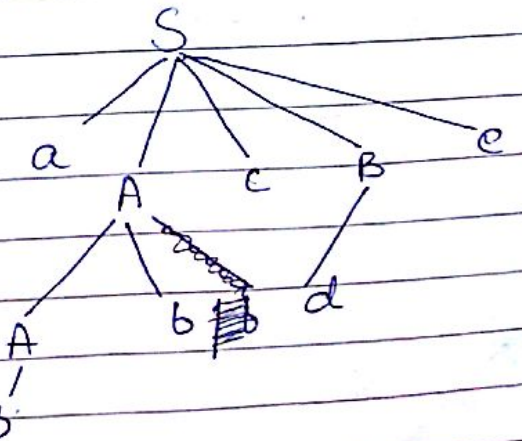
Right :-

$w = ibtibtaeaa$
$= ibtibtaeS$
$= ibtibtSeS$
$= ibtictSeS$
$= ibtS$
$= ictS$
$= S$

Eg :-

$S \rightarrow aAcBe$
$A \rightarrow Ab \mid b$
$B \rightarrow d$

$w = abbcde$
$= aAbcde$
$= aAcde$
$= aAcBe$
$= S$



Evergreen
Page No.
Date : / /201

Right Side

$w = abbcde$
$= abbcBe$
$= aAbcBe$
$= aAACBe$

$E \rightarrow E+E$
$E \rightarrow E*E$
$E \rightarrow id$

Left Sentential form

$E \rightarrow E+E$
$E \rightarrow E*E+E$
$E \rightarrow id*E+E$
$E \rightarrow id*id+E$
$E \rightarrow id*id+id$

Canonical form

$E \rightarrow E+E$
$E \rightarrow E+E*E$
$E \rightarrow E+E*id$
$E \rightarrow E+id*id$
$E \rightarrow id+id*id$

Handle, In stack handle is always on top of when right side derived from left hand side then will be no changes of ambiguity

id + id * id $

$ id
$ E+id
$ E+E*id
$ E+E*E
$ E+E
$ E

Four operation are used :-

1. Shift
2. reduce
3. Accept
4. Error

	Stack	Input	Action
(1)	$	id + id * id $	Shift id
(2)	$ id	+ id * id $	Shift +
(3)	$ id +	id * id $	

	Stack	Input	Action
(1)	$	id + id * id $	Shift id
(2)	$ id	+ id * id $	Reduce E → id
(3)	$ E	+ id * id $	Shift +
(4)	$ E +	id * id $	Shift id
(5)	$ E + id	* id $	Reduce E → id
(6)	$ E + E	* id $	Shift *
(7)	$ E + E *	id $	Shift id
(8)	$ E + E * id	$	Reduce E → id
(9)	$ E + E * E	$	Reduce E → E * E
(10)	$ E + E	$	Reduce E → E + E
(11)	$ E	$	Accept
	$ E	$	

Role of Stack in Shift reduce parsing

Boot Straping
Lexical Analysis → Role & need

Evergreen
Page No.
Date: / /201

$S \rightarrow S_1\ S_2\ S_3$

$S_1 \rightarrow E_3 + E_4 + E_5$

$E_1 \rightarrow a > b$

$E_2 \rightarrow b > c$

$S_2 \rightarrow$ if E_2 then S_3 else S_4

$S_3 \rightarrow E_6 + E_7$

$S_4 \rightarrow E_8$

Statement $\rightarrow$ if E_1 then S_1 else S_2