Lexical Analysis relates to DFA. (Later)

Examples -

(1) `+++-+--&&+;` No. of tokens = ?

8 tokens

(2) `;int a,b` tokens ?

= 5 tokens

This is not lexical error. It is syntax error & thus found in syntax analysis

(3) `for(i=0;i<=10;i++)` = 13 tokens

(4) `int 123;` = 3 tokens

(5) `printf("Hello, How are you ?");` = 5 tokens

(6)
```
if(a>b)
{
  a=b+c;
}
```
= 14 tokens

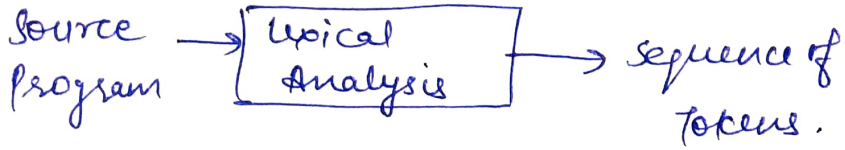
(7)
```
if fi(a>b)
{
  a=b+c;
}
```
Lexical Error

(8) `fi(a,b);` = 7 tokens (Here, fi is a function)

So, no error

CD (Theory)

1 Lexical Analysis - (Scanner)

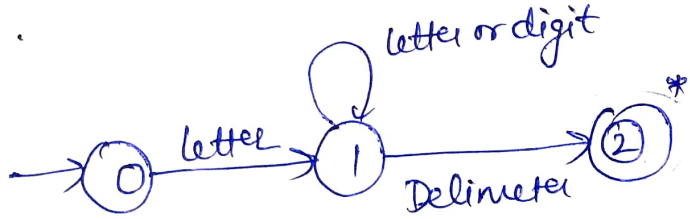


- Lexical Analyzers can be designed using Transition Diagrams (finite Automata).

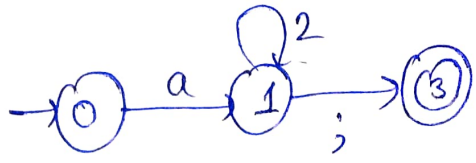
eg Transition diagram for 'if' keyword.

Transition diagram for an identifier -

An identifier starts with letter followed by letters or digits.



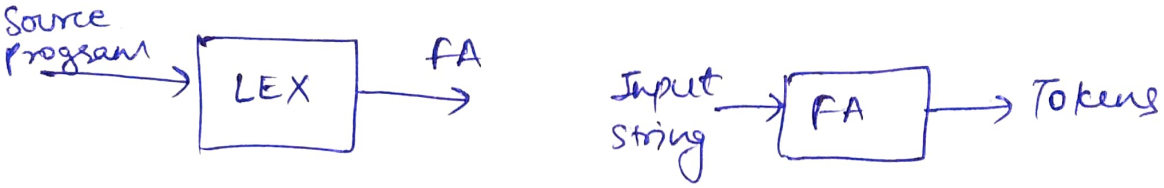
eg int a2; Here a2 is identifier

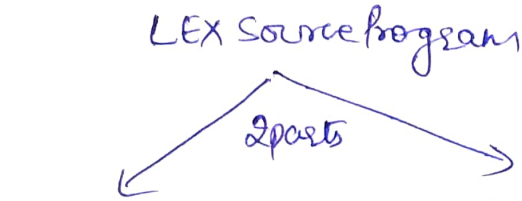


- Language for specifying Lexical Analyzer -

LEX source program represents the language used for specification of lexical Analyzer.

LEX - It is a tool/software which automatically generates lexical Analyzer / FA.



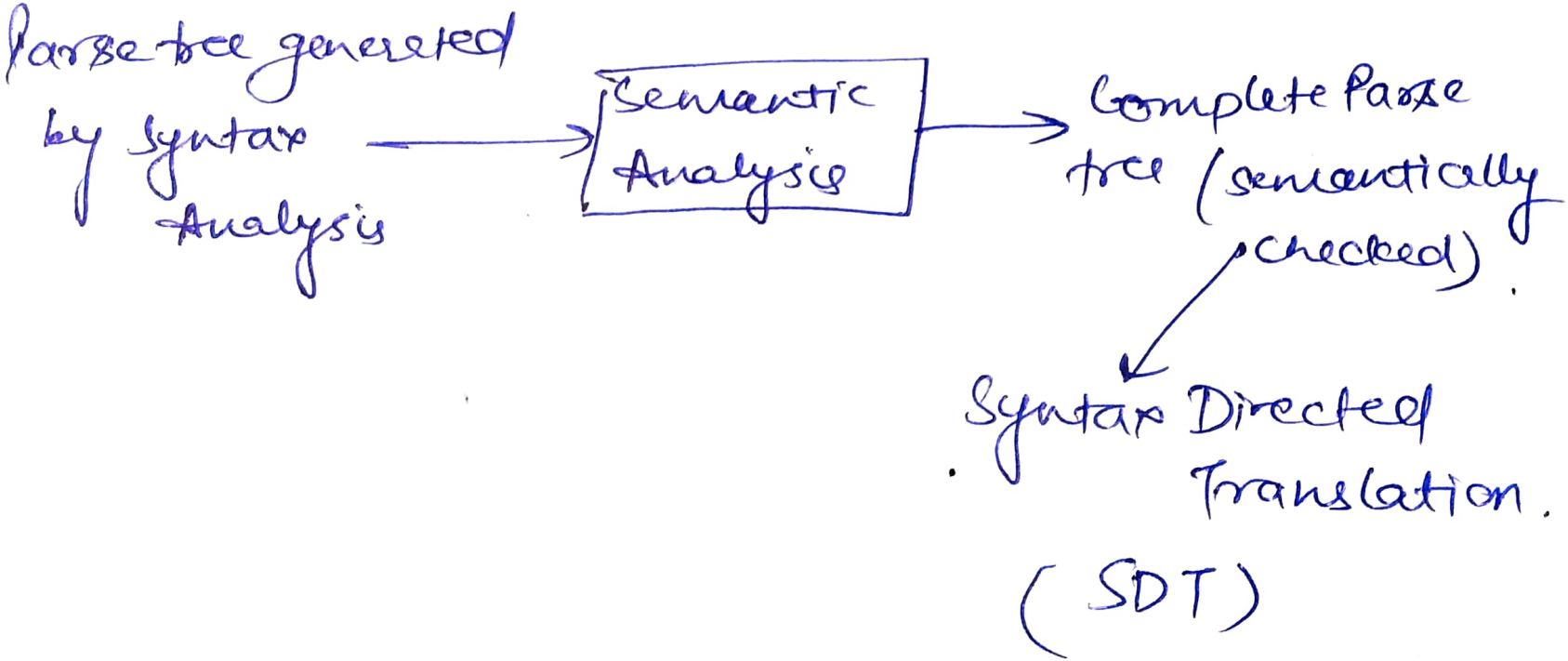


identifier = letter(letter/digit)*

↓
tells Lexical Analyzer to what it has to do on encountering the token.

(2) Syntax Analysis (Parser) -

(3) Semantic Analysis - (Type checking)



SDT - 2 schemes

Inter

Synthesized Translation

Value of variables on LHS of a production rule depend on the value of variables on RHS of production rule.

$E \rightarrow E_1 + E_2$

Inherited Translation.

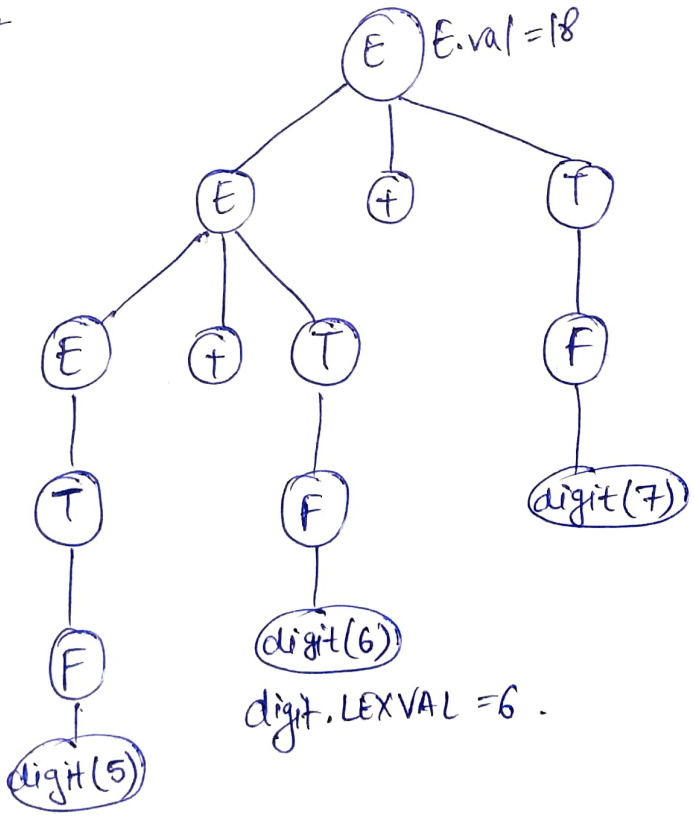
RHS depend on LHS

$A \rightarrow BC \quad \{B.val = 5 * A.val\}$

Annotated Parse Tree - A parse tree showing the values of attributes at each node.

LEXVAL represents the values of digits at leaf node.

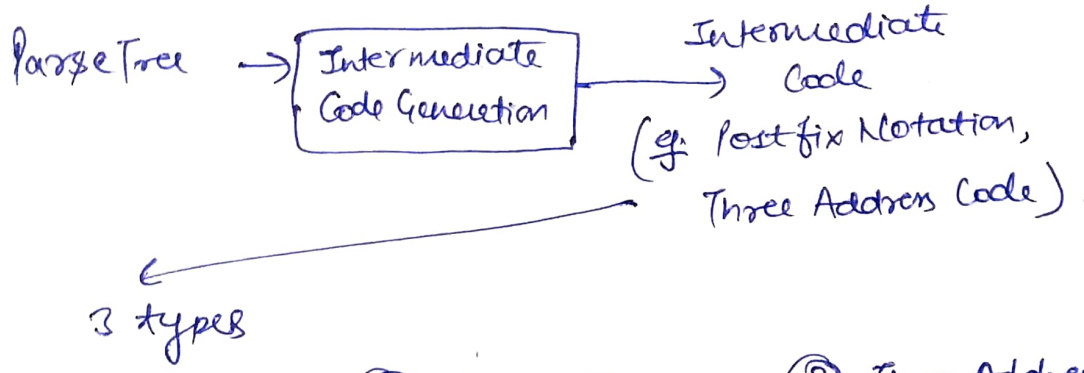
eg.



digit.LEXVAL = 6.

Tree for 5+6+7

Intermediate Code generation -

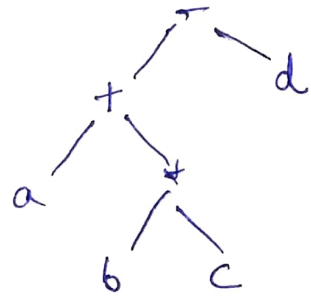


① Postfix Notation

(a+b) → ab+

② Syntax Trees

a+b*c-d



③ Three Address Code.

X = Y op Z

Result ↓ Operand1 ↓ Operand2

Operator.

① Quadruples (4 fields)

Operator	Argument 1	Arg 2	Result

② Triples (3 fields)

Operator	Arg 1	Arg 2

③ Indirect Triples.

Triples are indexed.

eg. a = b + c * d

↓ 3 address code

$t_1 = c * d$

$t_2 = b + t_1$

$a = t_2$.

(5) Symbol Table –

data structure that facilitates effective and effici way of storing information about various names appearing in source program.

Characteristics

(1) Entry is of the form –

Name	Info.
A	int
B	Real

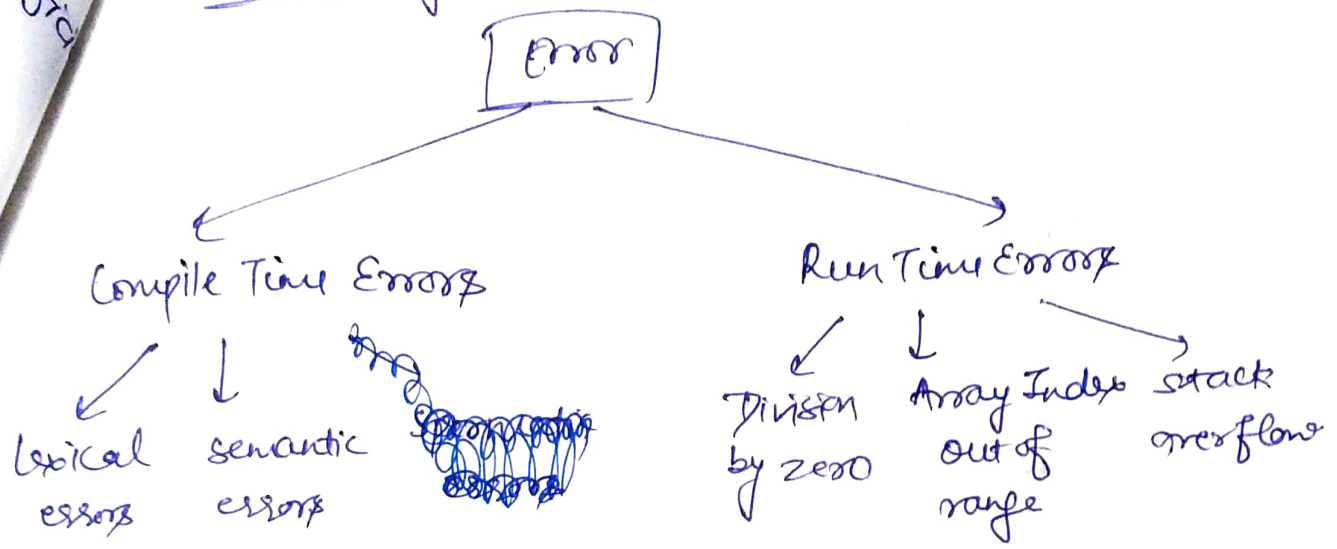
(2) Plays imp. role in each phase of compilation.

Operations – Searching, Editing, Deleting, Adding

Contents – 1. Variable Names
2. Constants
3. Procedure Names
4. function Names
5. Literal Constants & strings
6. labels
7. Compiler generated temporaries.

Data Structure 1. Lists
2. Self organization lists
3. Trees
4. Hash Tables.

Error Handling -



(7) Code Optimization - (optional phase)

(code improvement)

It refers to the techniques used by compiler to improve the execution efficiency of object code.

Classification of Code optimization

- Machine Dependent
  - e.g. Peephole Optimation
- Machine Independent.
  - Eliminate common sub-expressions
    - DAG is used (Directed Acyclic Graph)
  - Eliminate unreachable code

(8) Code Generation (final phase)

O/P is object code.

Types

- Absolute Code
  - ↓
  - m/c code that contain reference to actual address with in program's address space.
- Relocatable Machine code
  - ↓
  - relocatable object modules are linked together & loaded for execution with the help of linking loader
- Assembly code.
  - ↓
  - makes process of code generation easier.