Unit-1

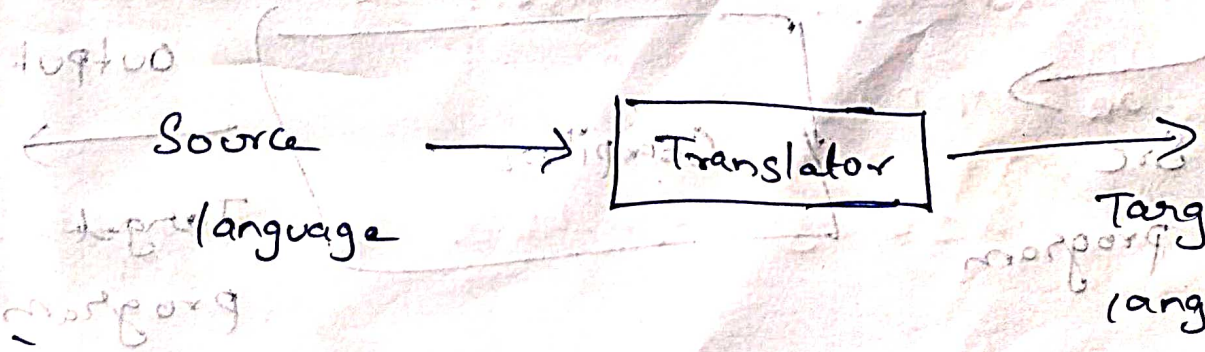
1.7 Translators:

* A translator is one kind of program that takes source code as as input and converts it into another form.

* The input program is called source program and the output is called target language.

* The various source languages are C, C++, Fortran, high level language.

* The target language can be low level language or a machine language.

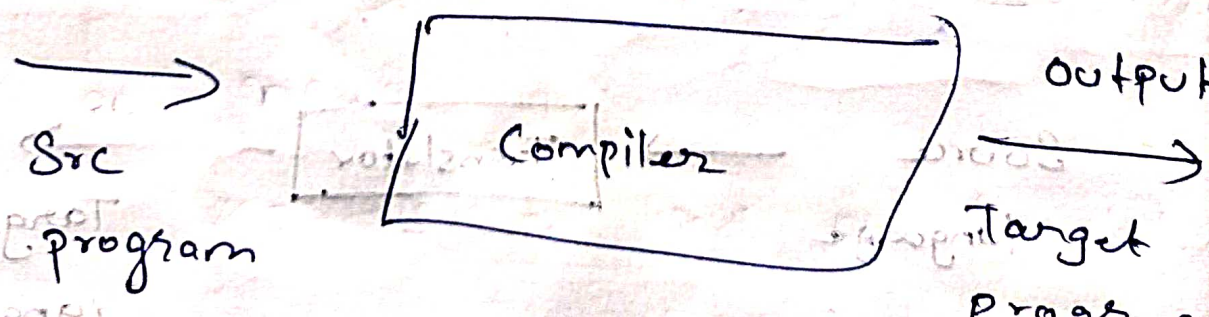


Compiler.

* A Compiler is a program which translates a high level language into low level or machine level languges.

* During this processs of translation if some errors are encountered then compiler displays them as error messages

* The compiler takes a source program as higher level languages such as C, Pascal, fortran



2) Language Processors :

* Language Processors are the cousins of Compiler.

* Compiler typically operates programs such as pre-processor, assemblers, loaders and link editors

1) Preprocessors :

* The output of preprocessors may be given as the input to Compiler.

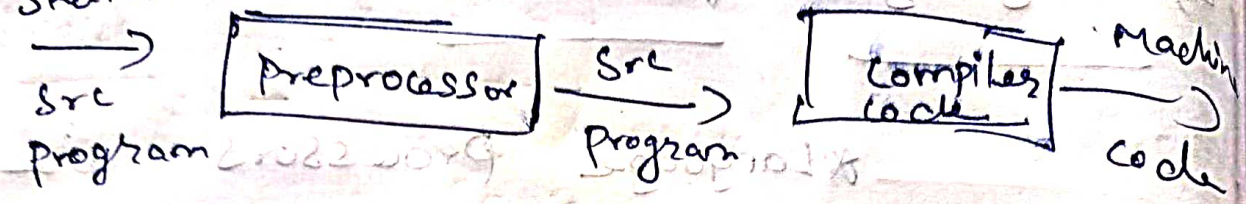
* preprocessors allows user to use macros in the program.

* Macros refers to repeated program

* preprocessors also includes header files.

eg:

```
#include <stdio.h>
```



Macro - Preprocessor:

* It is a type of small macro preprocessor.

* A small set of instructures

for eg:

#define PI 3.14

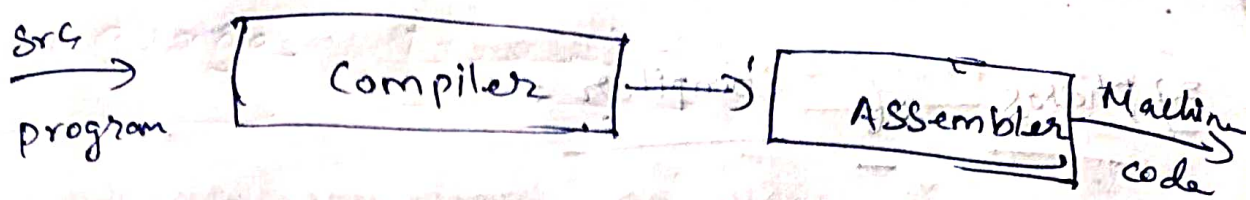
2) Assemblers:

* Some compilers produces assembly code as output.

* which is given to the assembler as an input.

* It is a kind of translator.

* Assembly code is mnemonic version of Machine code



* The instructions of Assembly code are binary language.

* The binary code is often called as Machine code.

* The machine code can be directly passed to loader / linker.

for eg:

```
MOV  a, R1
MUL  #5, R1
ADD  #7, R1
MOV  R1, b
```

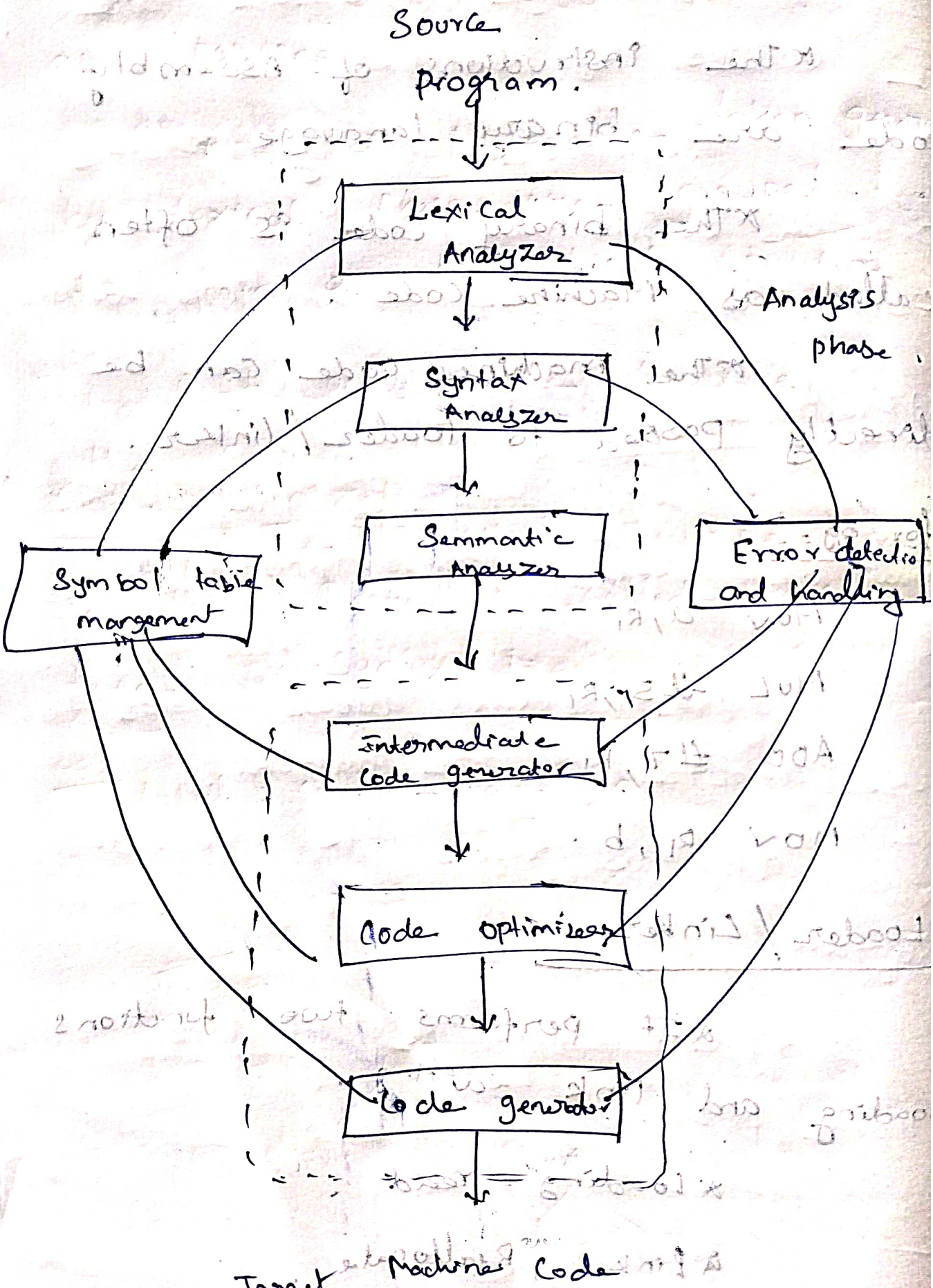
3) Loader / Linker

* It performs two functions loading and link editing.

* Loading - read.

* Link, Reallocate.

3) Phases of compiler :



* Lexical Analysis:

* Lexical analysis is also called scanning.

* In this phase the source code is scanned and broken up into group of strings called token.

* A token is sequence of character having a collective meaning.

* Blank statements are eliminated by Program

for eg :-

total = count + rate * 10 ;

total - identifier

= - assignment symbol

count - identifier

+ - plus sign

rate - Identifier

* - Multiplication sign

10 - constant number

Syntax Analysis.

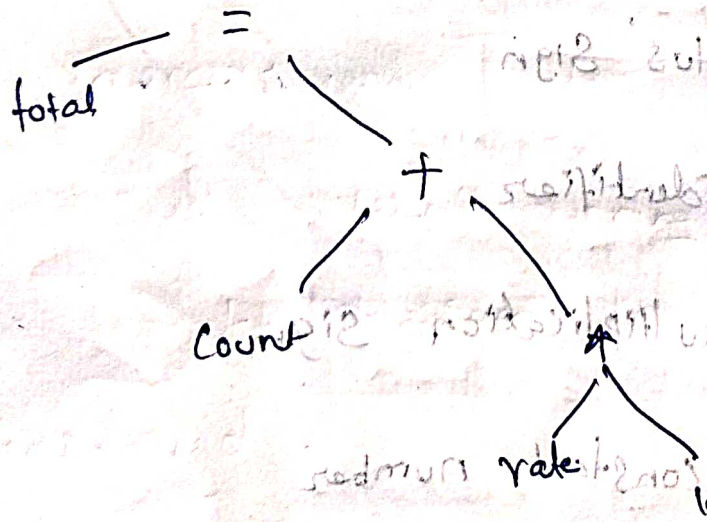
* The syntax analysis also called as parsing.

* In this phase the token generated by lexical phase are grouped together to form a hierarchical structure.

* It analysis the structure of the source string.

* The hierarchial structure generated in this phase is called parse tree or Syntax tree.

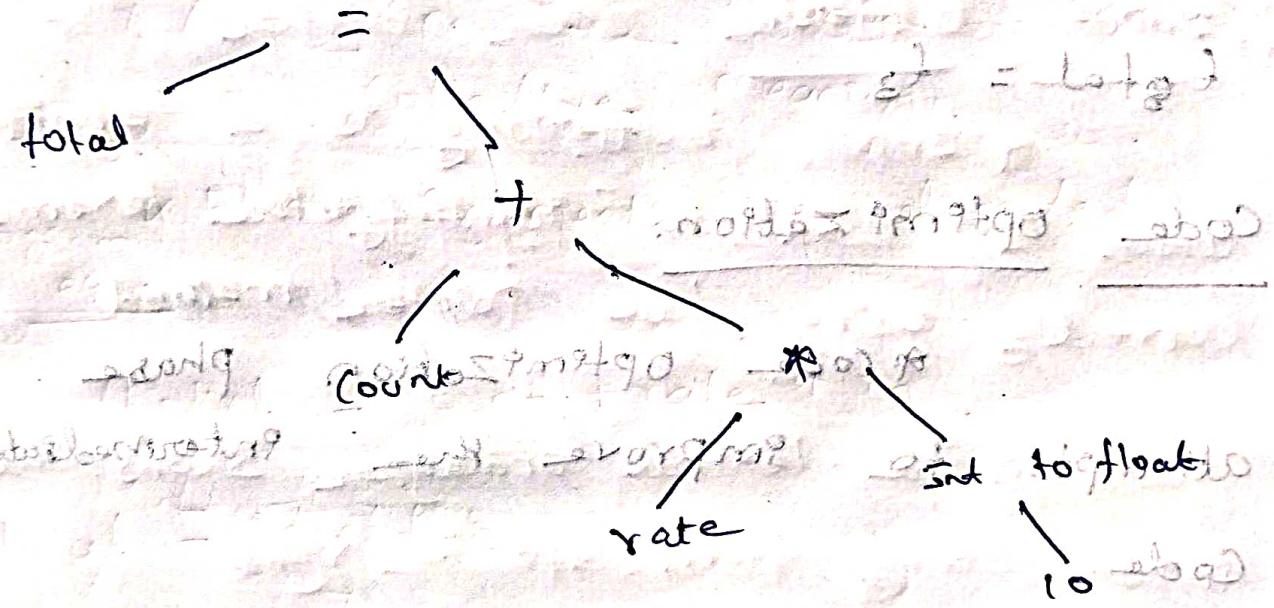
eg: total = count + rate * 10



Semantic Analysis:

* Once the Syntax is checked in the Syntax analyser phase the Next phase is Semantic analysis.

* It determines the meaning of the source string



Intermediate code generator:

* The intermediate code is a kind of code, which is easy to Generate.

* This code can be easily Converted into target code.

* There are three types of inter mediate code generator, such as three address code, quadruple, triple.

for example.

t_1 = int to float (10)

t_2 = rate * t_1

t_3 = t_2 * val = t_3

Code optimization:

* code optimization phase attempts to improve the intermediate code.

* This is neccessary to have a faster executing code or less consumption of memory.

* Thus overall running time of a target program can be improved.

Code Generator:

*In code generation phase the target code gets generated.

eg.

$a = b + c * 60$

↓

Lexical analyzer

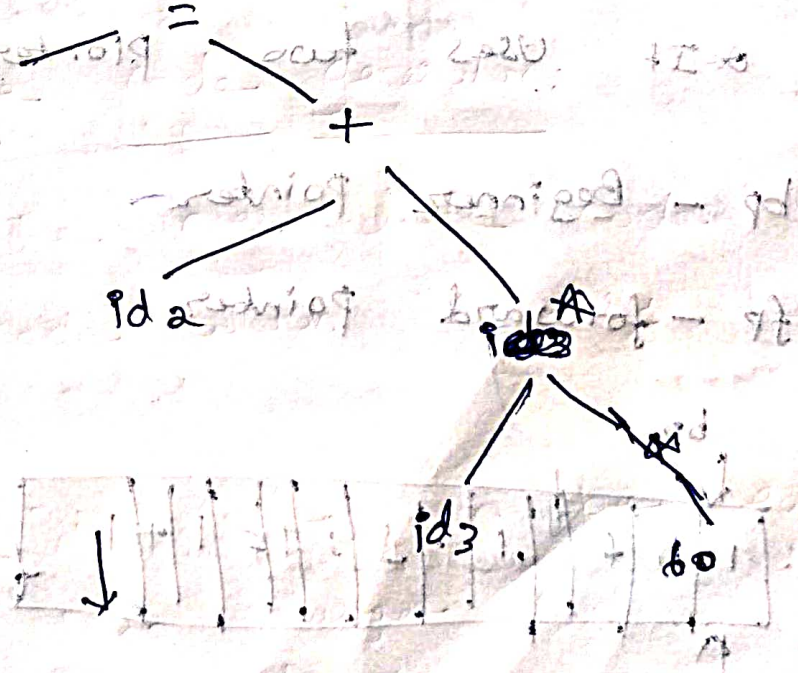
↓

$id_1 = id_2 + id_3 * 60$

↓

Syntax analyzer

↓



int to float

↓

Data Structure in compiler

* Stack
* Queue
* Trees
* Linked List
* DAG
* File

4) Input buffering.

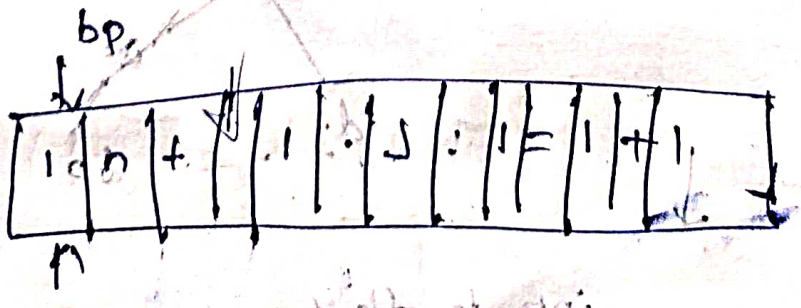
* Input bufferring is a technique which is used to identify the lexeme correctly using two pointer method

* The lexical analayzer scans the input string from left to right

* It uses two pointers name

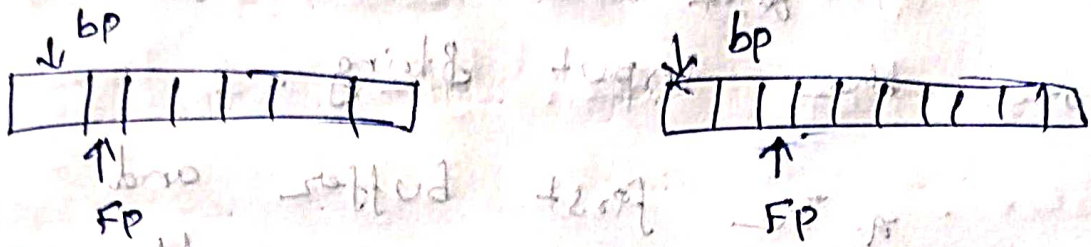
bp - Beginner pointer

fp - forward pointer



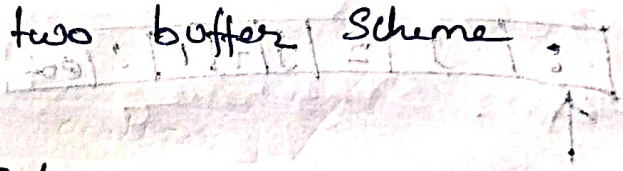
FP

* Initially both the pointer starts from first character



forward ptr() is used to moves ahead, and the blank spaces are encountered.

* There are two methods used in this context: they are one buffer scheme and two buffer scheme:



One buffer scheme:

* In this buffer only one buffer is used to store the input string.

* But the problem is lexeme is very large.

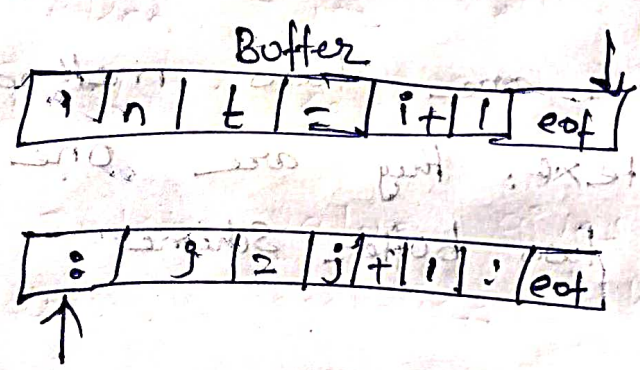
* It is an time consuming process.

2) Two buffer Scheme:

* It uses two buffer to Store the input string.

* The first buffer and second buffer Scanned alternately

*when end of current buffer is reached the other buffer is filled



Code:

```
if (fp == eof(buff1))
{
    fp++;
}
else if (fp == eof(buff2))
{
    fp++;
}
```

```
else if (fp == eof(input))
    return;
else
    fp++;
```

5) Specification of tokens:

*To Specify the tokens regular expressions are used.

*Regular expressions are mathematical Symbolisms.

*It provides convenient and useful notation for representing tokens

*Here is a set of rules denotes regular expression.

ε - empty string

$R = R_1 + R_2$ = Union operation

$R = R_1 . R_2$ = Concatenation operation

$R = R_1^*$ = Kleen Closure

6) Recognition of Tokens:

* Tokens are usually represented by a pair token type and token value.

* various type of tokens such as identifier, keywords, constants and operators and so on.

* Token Type tells us the category of token.

* Token value gives us the information regarding token.

* The token value is also called token attribute.

* During lexical analysis process the Symbol table is maintained.

for eg:

Token	code
if	1
else	2

while 3

for 4

identifier 5

Constant 6

6. Finite Automata:

A finite automata is a collection of 5 tuples $(Q, \Sigma, \delta, q_0, F)$

$Q \rightarrow$ Set of finite States

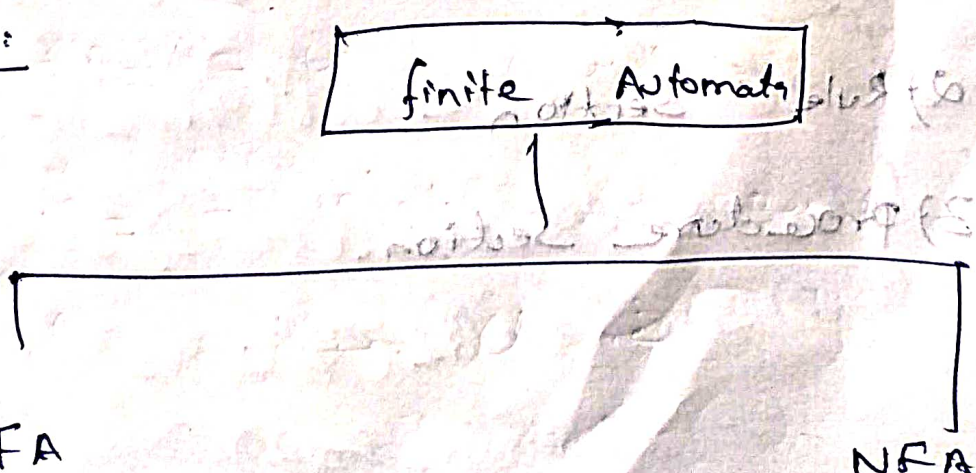
$\Sigma \rightarrow$ Input alphabets

$q_0 \rightarrow$ Initial state

$f \rightarrow$ final state

$\delta \rightarrow$ Transition function.

Types:



7) Lex tool

* for effecient design of Compiler various tools have been used for the construction of lexical analyzer

* Basically LEX is a unix Utility generates the lexical analyzer.

* LEX is very much faster in findings the tokens.

* The Lex program is handwritten in C.

* LEX scans the source program to get the stream of tokens.

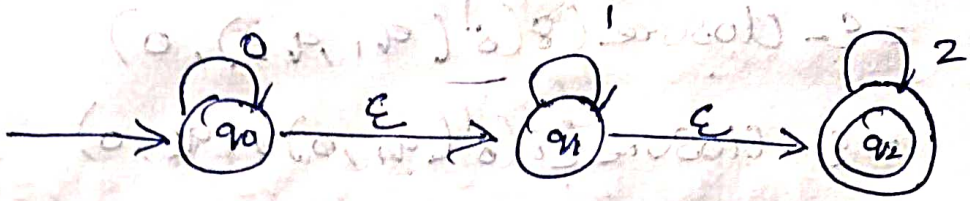
Structure of a LEX:

1) Declaration Section

2) Rule Section

3) Procedure Section

1. NFA with ϵ to without ϵ



ϵ-closure $q_0 = \{q_0, q_1, q_2\}$

ϵ-closure $q_1 = \{q_1, q_2\}$

ϵ-closure $q_2 = \{q_2\}$

$\delta'(q_0, 0) = \epsilon\text{-closure}(\delta(\delta'(q_0), 0))$

$= \epsilon\text{-closure}(\delta(\delta'(q_0, q_1, q_2), 0))$

$= \epsilon\text{-closure}(\delta(\delta'(q_0, 0) \cup (q_1, 0) \cup (q_2, 0)$

$= \epsilon\text{-closure}(q_0 \cup \phi \cup \phi)$

$= \epsilon\text{-closure}(q_0)$

$= \epsilon\text{-closure}(q_0, q_1, q_2)$

$\delta'(q_0, 1) = \epsilon\text{-closure}(\delta(\delta'(q_0), 1))$

$= \epsilon\text{-closure}(\delta(\delta'(q_0, q_1, q_2), 1))$

$= \epsilon\text{-closure}(\delta(\delta'(q_0, 1) \cup (q_1, 1) \cup (q_2, 1))$

$= \epsilon\text{-closure}(\phi \cup q_1 \cup \phi)$

$= \epsilon\text{-closure}(q_1) = (q_1, q_2)$

$\delta'(q_1, 0) = \varepsilon\text{-closure}(\delta(\delta'(q_1), 0)$

$= \varepsilon\text{-closure}(\delta(\delta'(q_1, q_2), 0)$

$= \varepsilon\text{-closure}(\delta(\delta'(q_1, 0) \cup (q_2, 0)$

$= \varepsilon\text{-closure}(\delta(\delta'(\phi) \cup \phi$

$= \varepsilon\text{-closure}(\phi \cup \phi)$

$= \varepsilon\text{-closure}(\phi)$

$= \phi$

$\delta'(q_1, 1) = \varepsilon\text{-closure}(\delta(\delta'(q_1), 1)$

$= \varepsilon\text{-closure}(\delta(\delta'(q_1, q_2], 1)$

$= \varepsilon\text{-closure}(\delta(\delta'(q_1, 1) \cup (q_2, 1)$

$= \varepsilon\text{-closure}(q_1] \cup (\phi)$

$= \varepsilon\text{-closure}(q_1)$

$= \{q_1, q_2\}$.

$\delta'(q_1, 2) = \varepsilon\text{-closure}(\delta(\delta'(q_1), 2)$

$= \varepsilon\text{-closure}(\delta(\delta'(q_1, q_2), 2)$

$= \varepsilon\text{-closure}[\delta(\delta'(q_1, 2) \cup (q_2, 2)$

$= \varepsilon\text{-closure}(\phi \cup q_2)$

$= \varepsilon\text{-closure}(q_2)$

$= \{q_2\}$

$\delta'(q_2, 0) = \varepsilon\text{-closure}(\delta(\delta'(q_2), 0)$

$= \varepsilon\text{-closure}\,\delta'(q_2, 0)$

$= \varepsilon\text{-closure}(\phi)$

$= \varepsilon\text{-closure}(\phi)$

$= \phi$

$\delta'(q_2, 1) = \varepsilon\text{-closure}(\delta(\delta'(q_2), 1)$

$= \varepsilon\text{-closure}\,\delta'(\delta'(q_2, 1)$.

$= \varepsilon\text{-closure}(\phi)$

$= \varepsilon\text{-closure}(\phi)$

$= \phi$

$\delta'(q_2, 2) = \varepsilon\text{-closure}(\delta(\delta'(q_2), 2)$

$= \varepsilon\text{-closure}(\delta(\delta'(q_2, 2)$

$= \varepsilon\text{-closure}[\delta(\delta'(q_2)$

$= \varepsilon\text{-closure}(q_2)$

$= q_2$

$\delta'(q_0, 2) = \varepsilon\text{-closure}(\delta(\delta'(q_0), 2)$

$= \varepsilon\text{-closure}(\delta(\delta'(q_0, 2)$

$= \varepsilon\text{-closure}(q_0, q_1, q_2), 2)$

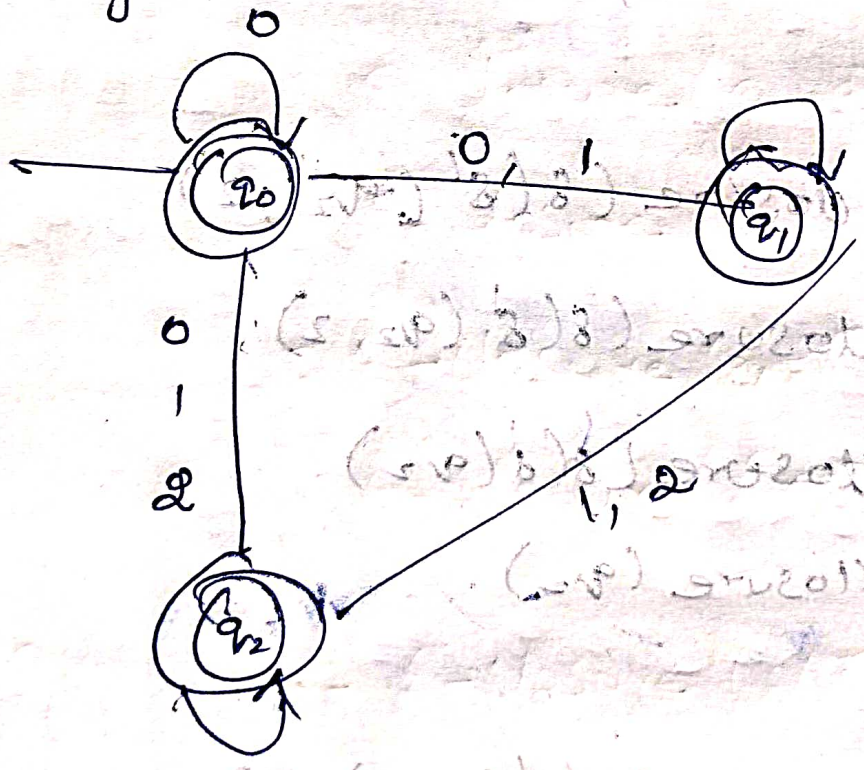
$= \varepsilon\text{-closure}(q_0, 2) \cup (q_1, 2) \cup (q_2, 2)$

$= \varepsilon\text{-closure}(q_2, 2)$

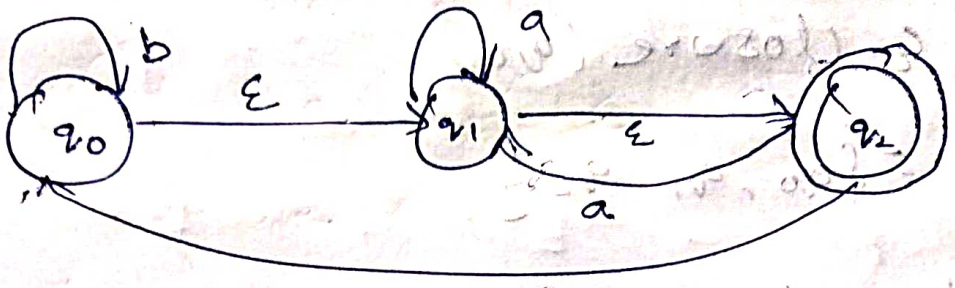
$= \varepsilon\text{-closure}(q_2) = q_2$

Input / State	0	1	2
q_0	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
q_1	ϕ	$\{q_1, q_2\}$	$\{q_2\}$
q_2	ϕ	ϕ	$\{q_2\}$

1. Diagram



Equivalence of NFA with ε to DFA



ε-closure $(q_0) = \{q_0, q_1, q_2\}$

ε-closure $(q_1) = \{q_1, q_2\}$

ε-closure $(q_2) = \{q_2\}$

Let call q_0 as a A

$\delta'(A, a) = \varepsilon\text{-closure}(\delta'(A), a))$

$= \varepsilon\text{-closure}\,\delta(\delta'(q_0, q_1, q_2), a)$

$= \varepsilon\text{-closure}\,(q_0, a) \cup (q_1, a) \cup (q_2, a)$

$= \varepsilon\text{-closure}\,(\phi) \cup (q_1) \cup \phi$

$= \varepsilon\text{-closure}\,\{q_1\}$

$= \{q_1, q_2\}$ Let call this as B

$\delta'(A, b) = \varepsilon\text{-closure}(\delta'(A), b))$

$= \varepsilon\text{-closure}\,\delta(\delta'(q_0, q_1, q_2), b)$

$= \varepsilon\text{-closure}\,\delta(\delta'(q_0, b) \cup (q_1, b) \cup (q_2, b)$

ε-closure $\delta(\delta'(q_0) \cup \phi \cup \phi$

ε-closure (q_0)

$= \{q_0, q_1, q_2\}$

$\delta'(A, a) = B$

$\delta'(A, B) = A$

$\delta'(B, a) = \varepsilon$-closure $(\delta(\delta'(B), a)$

$= \varepsilon$-closure $(q_1, q_2), a)$

$= \varepsilon$-closure $(q_1, a) \cup (q_2, a)$

$= \varepsilon$-closure $(q_1) \cup \phi$

$= \varepsilon$-closure (q_1)

$= \varepsilon$-closure $\{q_1, q_2)$

$\delta'(B, b) = \varepsilon$-closure $(\delta(\delta'(B), b)$

$= \varepsilon$-closure $(\delta(\delta'(q_1, q_2), b)$

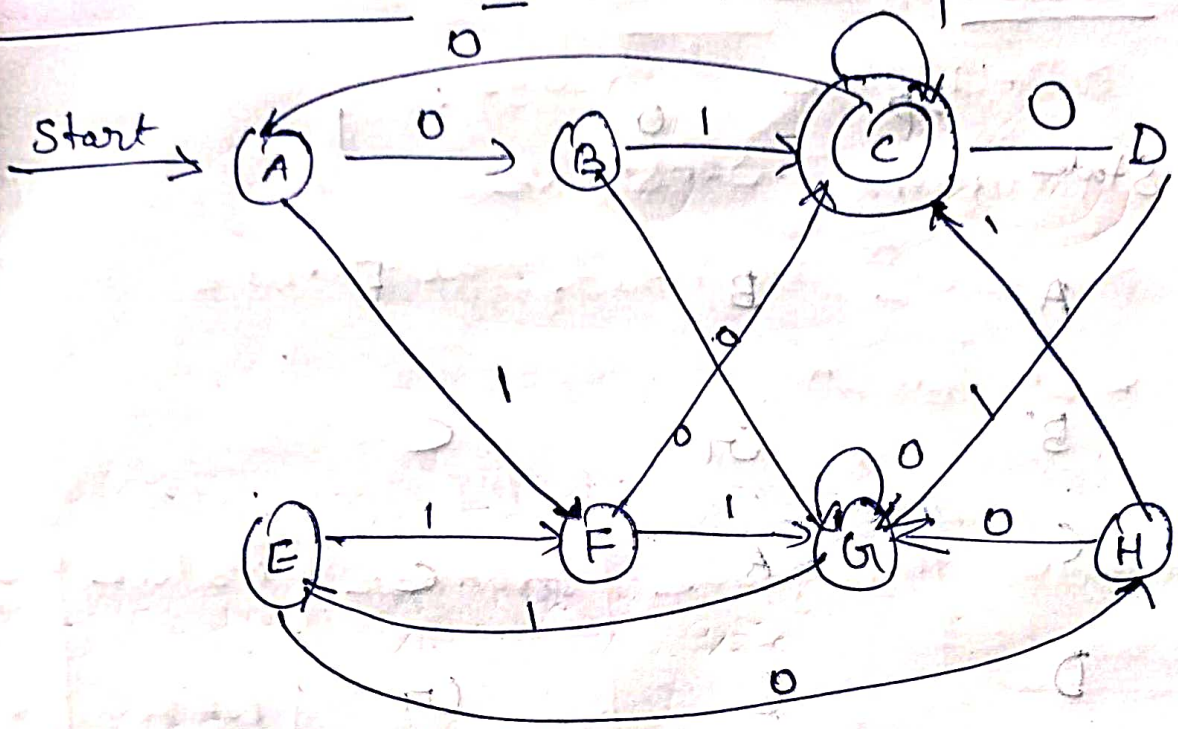
$= \varepsilon$-closure $(q_1, b) \cup (q_2, b)$

$= \varepsilon$-closure $(\phi) \cup \phi$

$= \varepsilon$-closure (ϕ)

$= \phi$

Minimization of DFA



I/P State	0	1
A	B	F
B	G	C
(C)	A	C
D	C	G
E	H	F
F	C	G
G	G	E
H	G	C

Minimized DFA