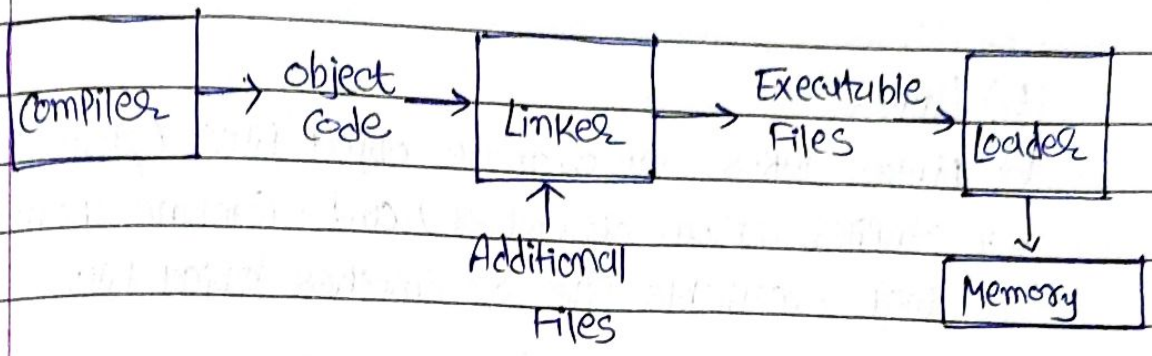
(*)



(*) Meanings of :-

1) High level language (User oriented language)

↳ A High level language is any Programming language that enables development of a program in a much more user-friendly programming context & it is independent of the Computer's hardware architecture.

↳ C, C++, Java is High level language.

2) Assembly language

↳ assembly language is a low-level programming language that is intended to communicate directly with a Computer's hardware.

3) Machine level language

↳ low-level language is the only language which can be understood by the Computer. low-level language is also known as Machine language.

4) Translator

↳ A translator is a Program that converts source code into machine code.

5) Linker :-

↳ Linker takes one or more object files (generated by a compiler or an assembler) and combine them into a single executable file or another object file.

6) Loader :-

↳ It is special program that takes input of executable files from the linker, loads it to main memory & prepare this code for execution by Computer.

(*) Real time application of translators :-

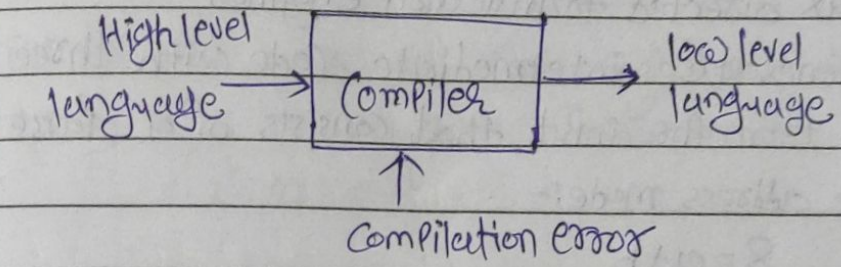
(*) Computer → Hardware + Software

Piece of Mechanical Device

understand instruction in the form of electronic charge which is the counter part of binary language in software

↳ It is function are being controlled by Compatible Software

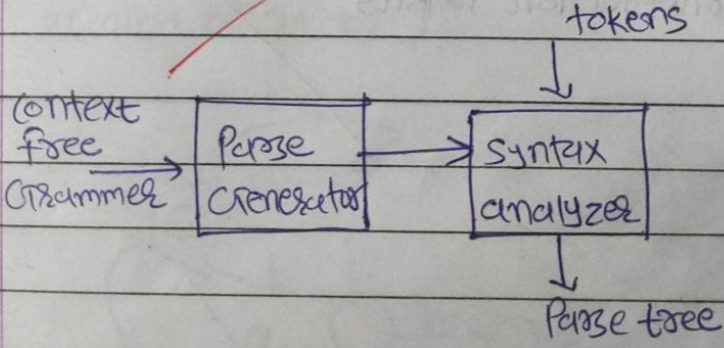
LT LAB Aim:- Study the Compiler's & various Selective tools.



↳ A Compiler is a special Program that transcates a Programming language's source code into machine code, bytecode or another Programming language.

Various Selective tools of Compiler's :-

1) Parse Generator:-



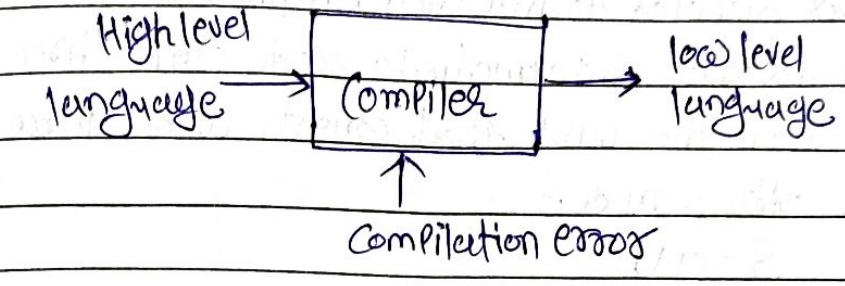
(*) Computer → Hardware + Software

Piece of Mechanical Device

understand instruction in the form of electronic charge which is the counter part of binary language in software

↳ It is function are being controlled by compatible software.

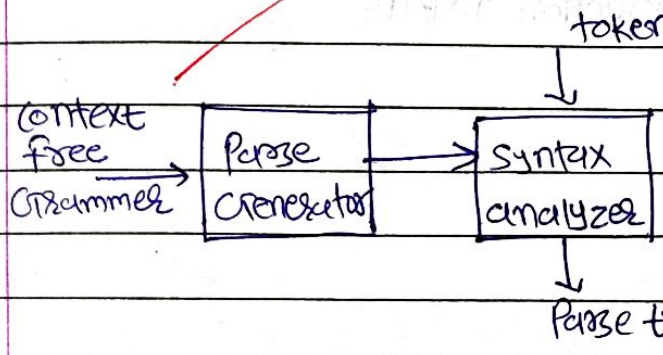
LT LAB Aim:- Study the Compiler's & various selective tools.



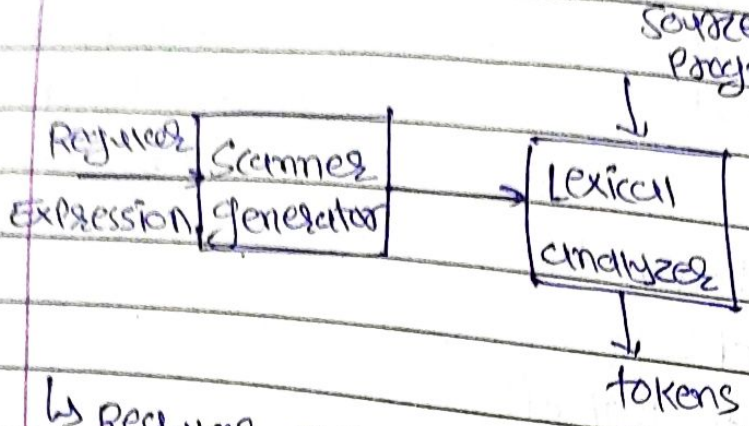
↳ A Compiler is a special program that translates a programming language's source code into machine code, bytecode or another programming language.

Various selective tools of Compiler's :-

1) Parse Generator:-



2) Scanner generators :-



↳ Regular Expression ला FA generate करतो.
↳ lexical analyzer is responsible for removing the white spaces & comments from the source program

3) Syntax directed translation engines
↳ It generates intermediate code with three address format from the input that consists of a parse tree.
Three address mode :-
$x = a + b$
a+b ची value x ला store करतो.

4) Automatic code generators (input as a optimization code & generate machine code)

5) Data-flow analysis engines (to make our code as fast as possible / code optimization)

6) Compiler Construction toolkits

(*) What do you mean by Regular expression ?
=> A language whose DFA is possible.

1) Write the RE (Regular expression) for the language accepting all the string which are starting with 1 & ending with 0.

$1(0+1)^*0$

2) RE for the language starting with a but not having consequitive b's.

$a(ba+a)^*(a+b)$

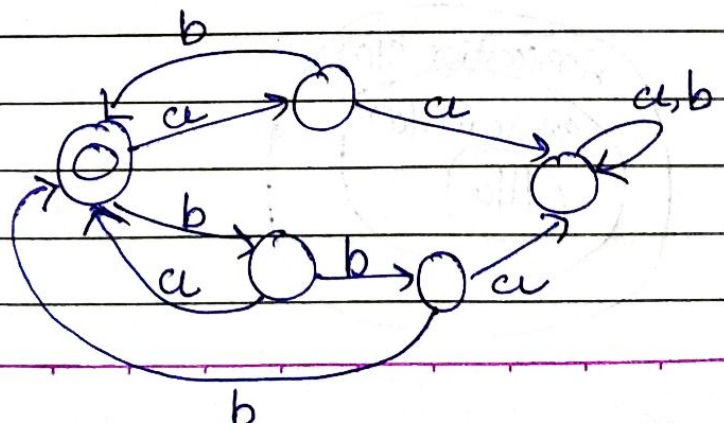
3) RE for the language containing the string in which every 0 is immediately follow by 11.

$(011+1)^*$

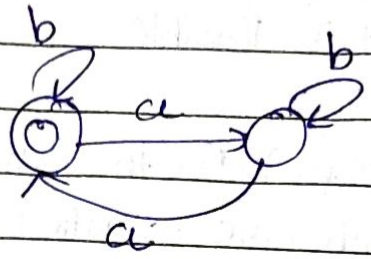
(*) Difference between Compiler & interpreter
Difference between CFG & RE
Difference between linker & loader

(*) Design FA :-

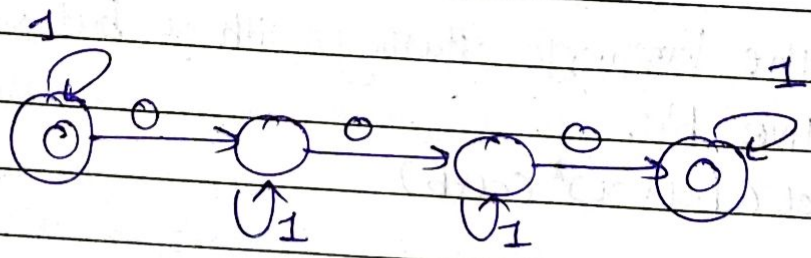
1) $(ab+b(a+bb))^*$



2) $(b+ab^*a)^*$



3) $1^*(01^*01^*01^*)$

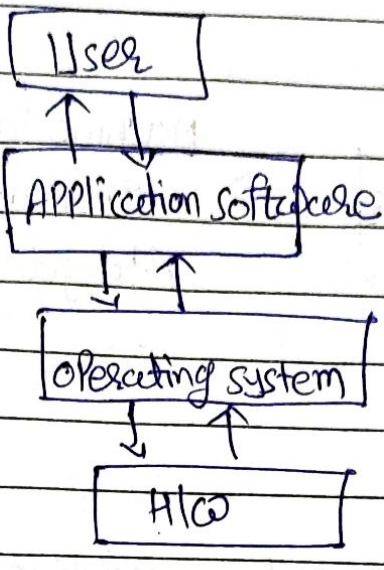


Chapter language translation overview

(*) Will & Component of system :-

Application Software :- Word, excel, browser, bank Manage system, Accounting software

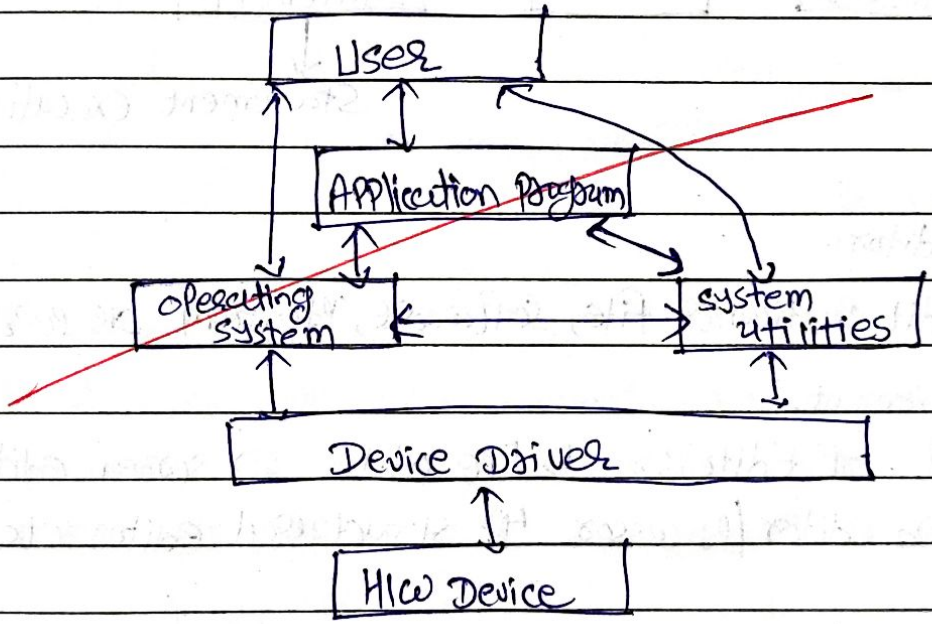
System Software :- OS, Compiler, Assembler, Interpreter



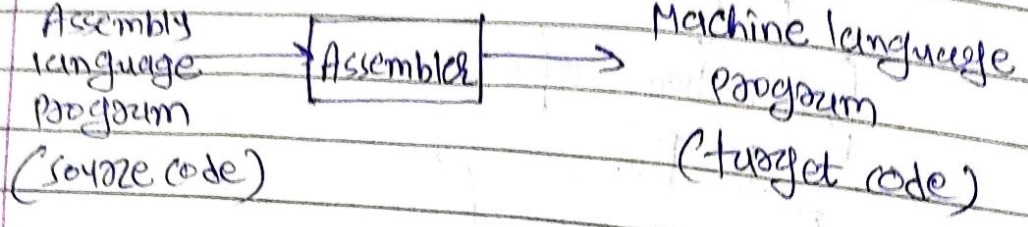
System S/w :-

- Is manage basic function such as storing Data, Retriving files & Scheduling task, etc.
- Is to make effective execution of each program, to make effective utilization of all the resources.

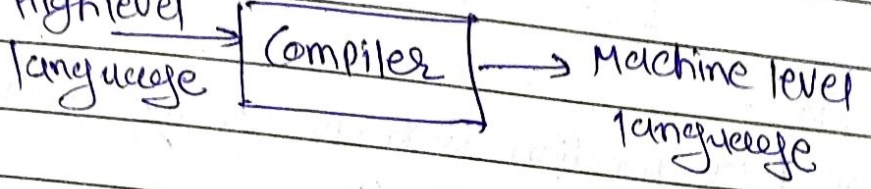
(*) Component of system S/w :-



1) Assembler :-

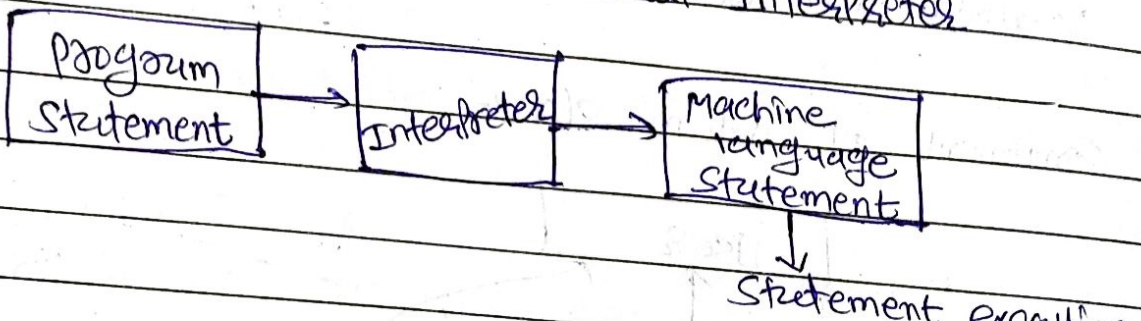


2) Compiler :-



3) Interpreter :-

↳ line by line execution करने वाला Interpreter

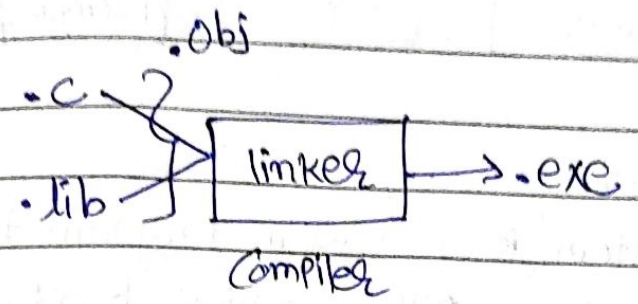


4) Editor :-

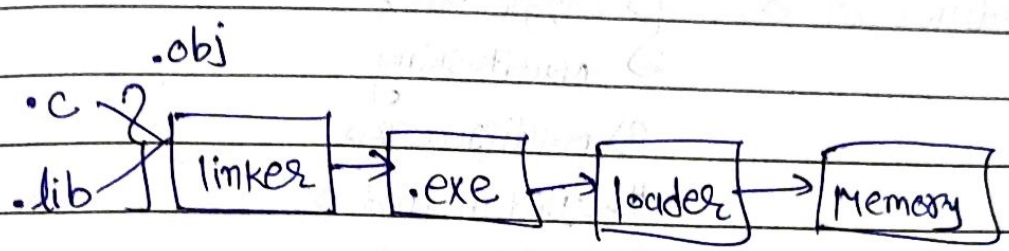
↳ Edit particular file, software, program as per the Requirement.

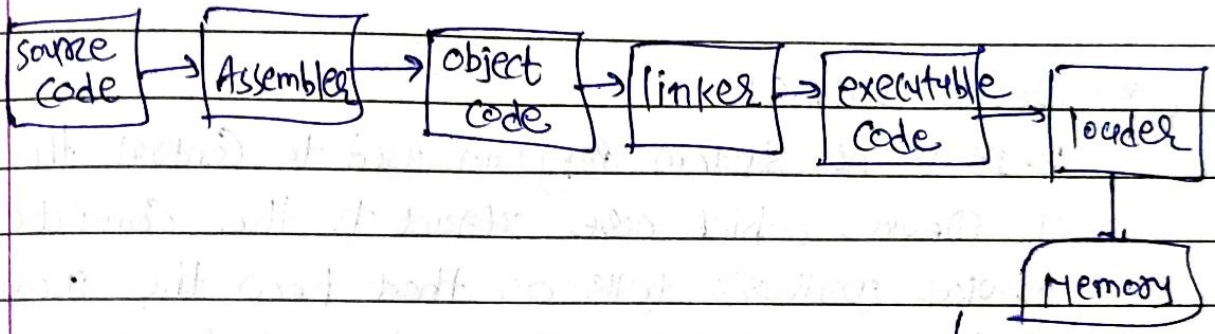
↳ types of Editors :- 1) line editor 2) screen editor 3) word editor / processor 4) structural editor etc.

5) linker :-



6) loader :-





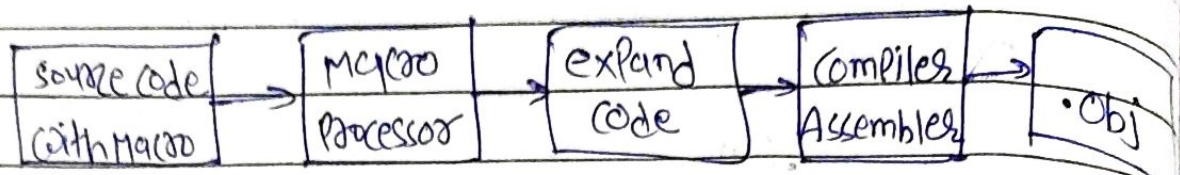
↳ object program ready for execution

7) Debugger :-

↳ A Debugger is a Computer program used to find out the errors which are also called as bugs in source program.

8) MACRO :-

↳ Code Reuselibility



(*) Operating System:-

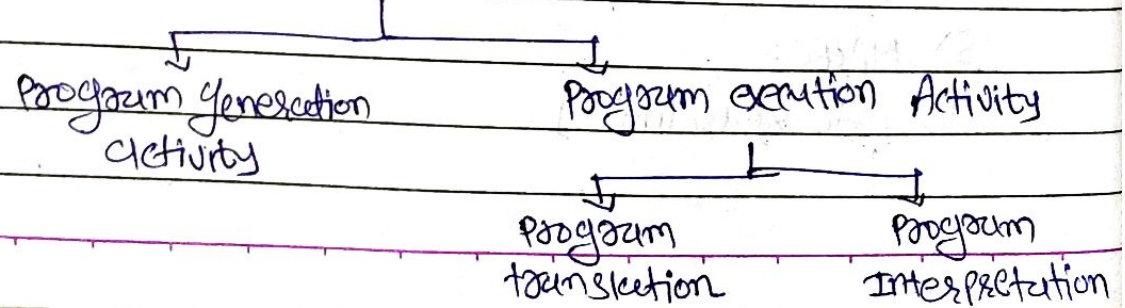
- Operating system is a system program that enables a user to communicate with Computer hardware
- linux, windows, unix is a example of OS
- types of os:- 1) Single user
  2) Multitasking
  3) Multiprocessor
  4) Distributed os
  5) Network os

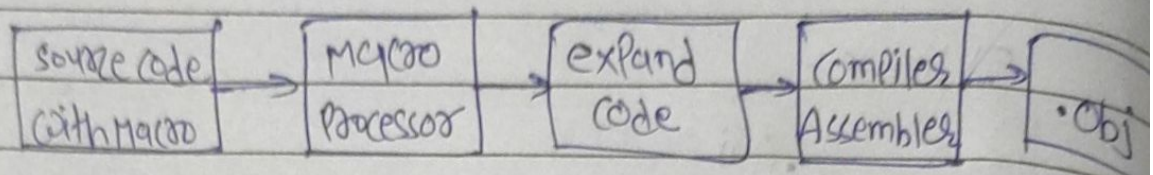
(*) Device Driver:-

- It is a system program used to control the no. of Devices which are attact to the Computer. Device Driver's tells os that how the Device will work or certain Commands which are generated by the user.
- Example:- Mouse Driver, keyboard Driver, Wifi Driver

(*) language Processing activities:-

- language Processor:- Convert Source code into Machine code





(*) Operating System:-

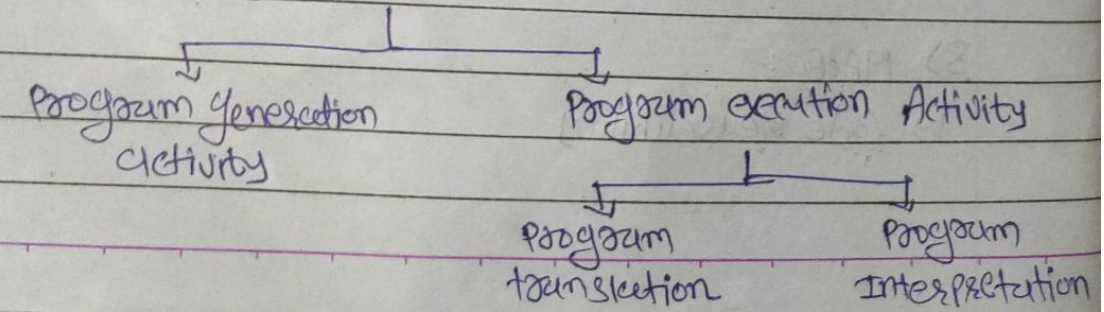
- Operating system is a system program that enable a user to communicate with Computer hardware
- linux, window, unix is a example of OS
- types of OS:- 1) Single user
  2) Multitasking
  3) Multiprocessor
  4) Distributed OS
  5) Network OS

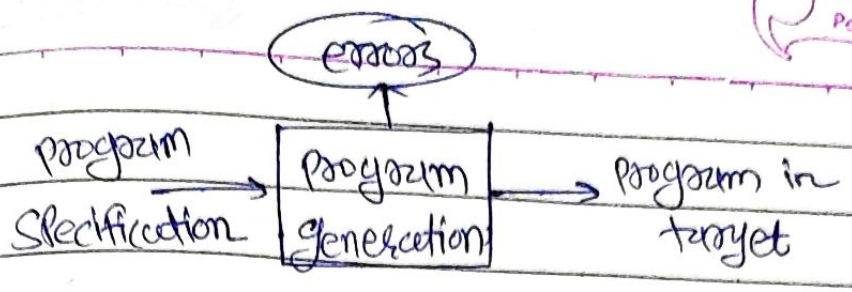
(*) Device Driver:-

- It is a system program used to control the no. of Devices which are attact to the Computer. Device Driver's tells OS that how the Device will work or certain Commands which are generate by the user.
- Example:- Mouse Driver, keyboard Driver, Wifi Driver

(*) language Processing activities:-

- language Processor:- Convert Source code into Machine code

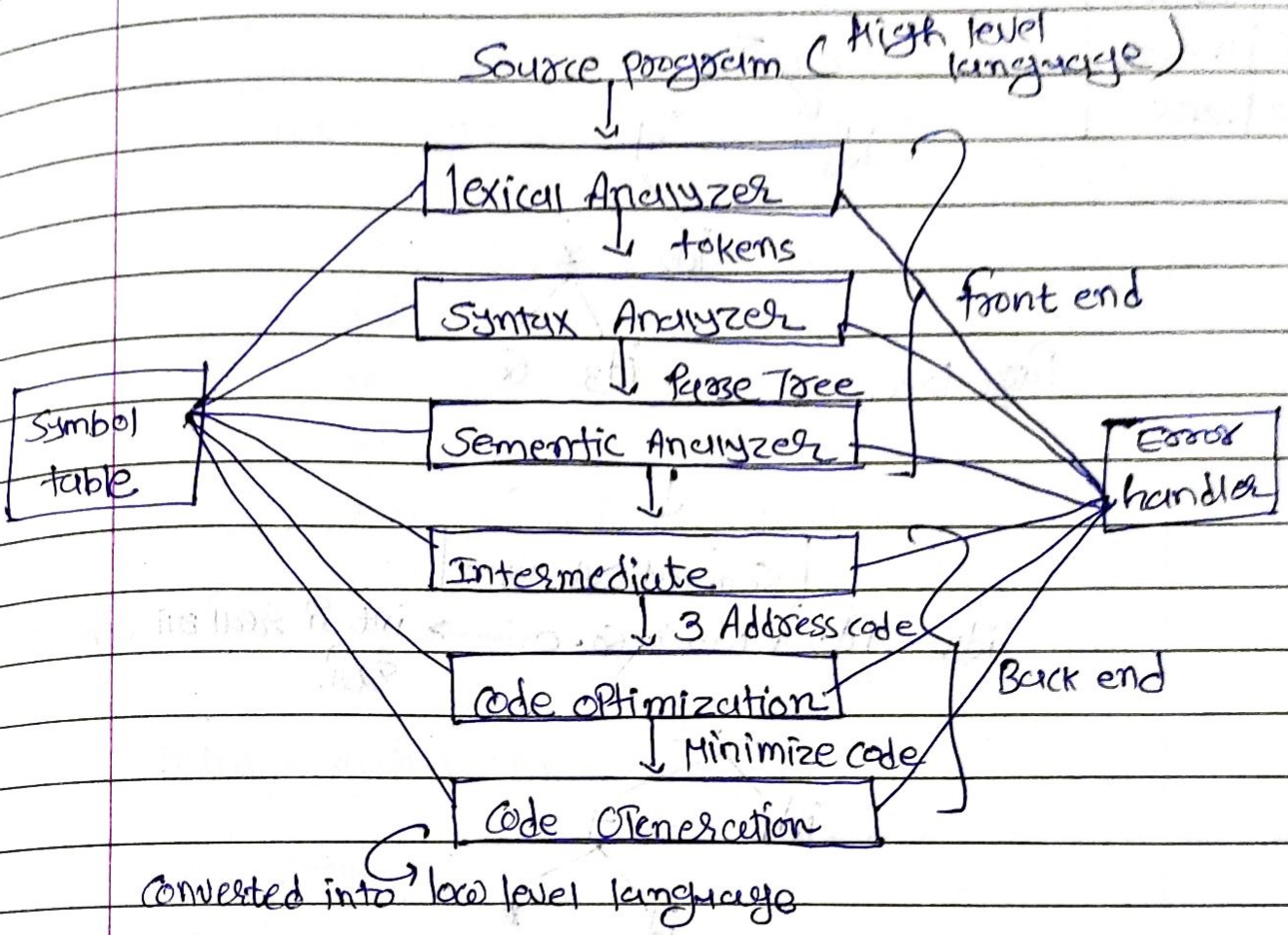



errors
program
specification
program
generation
program in
target

H.W.
Total = a/b+c*60

13/12/2022

IMP (*) phases of Compiler :



(*) Position := initial + rate * 60

lexical Analyzer

Position → identifier
:= → Assignment operator
initial → identifier
+ → operator
rate → identifier
* → operator
60 → constant (int value)

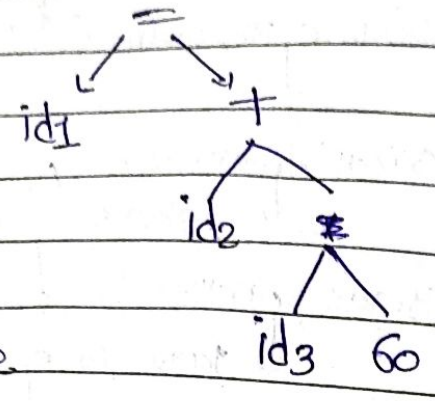
$id_1 = id_2 + id_3 * 60$

इसको token में convert करेगा.

Symbol Table

1	Position
2	initial
3	rate

Syntax Analyzer

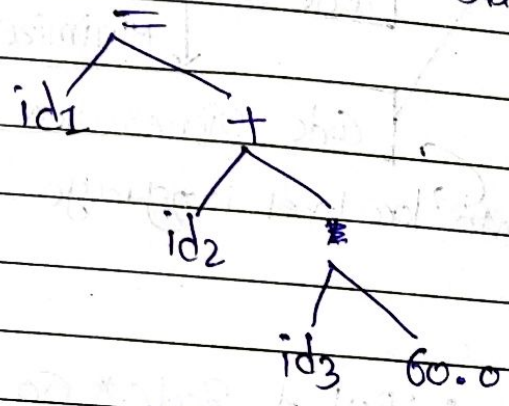


Parse tree

↓

Semantic Analyzer

$id_1 = id_2 + id_3 * 60.0$ → int ला real मा convert गर्छ।



↓

Intermediate code

temp1 = int to real (60)
temp2 = id3 * temp1
temp3 = id2 + temp2
id1 = temp3

↓

Code optimization

```
temp2 = id3 * 60.0
id1 = id2 + temp2
```

} Minimum variable નો use કરવાની trey કરવું જેથી code optimization કહેવું.

↓

Code generation

```
MOVF R1, id3
MULF R1, #60.0
MOVF R2, id2
ADDF R1, R2
MOVF id1, R1
```

1) total := a|b + c*60

lexical Analyzer

total

:=

a

|

b

+

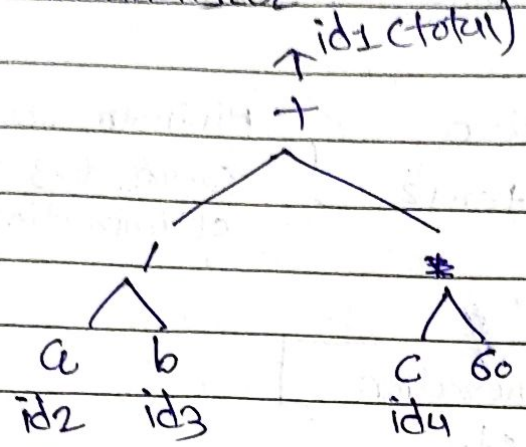
c

*

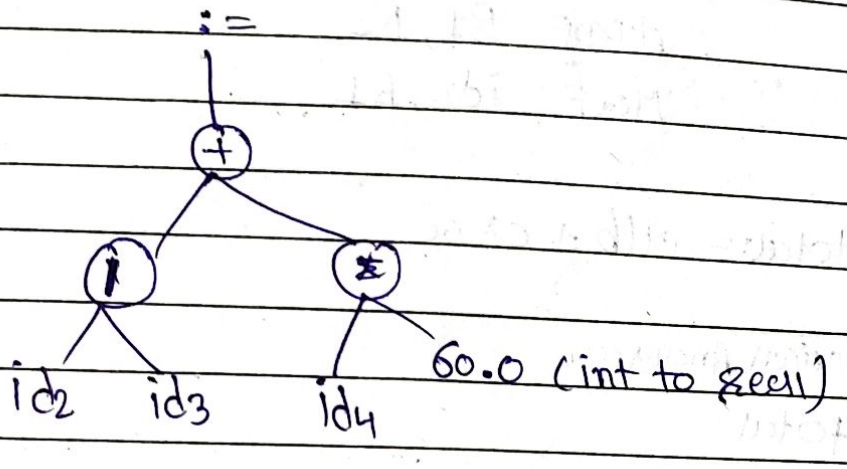
60

id1 := id2 | id3 + id4 * 60

Syntax Analyzer



Semantic Analyzer



intermediate code:-

```
temp1 = a/b
temp2 = int to Real(60)
temp3 = c * temp2
total = temp1 + temp2
```

Code optimizer:-

```
temp1 = a/b
temp2 = c*60
id1 = temp1 + temp2
```

temp1 = id2 / id3

temp2 = id4 * 60

id1 = temp1 + temp2

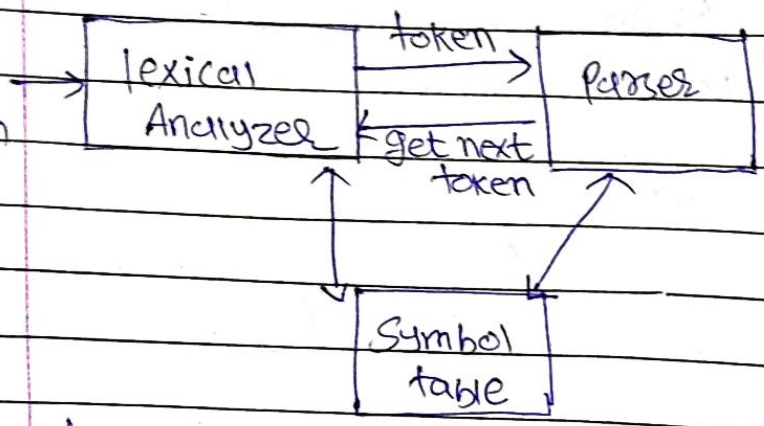
Code generator :-

```
MOVF R1, id2
DIVF R1, id3
MOVF R2, id4
MULF R2, 60
ADDF R1, R2
MOVF id1, R1
```

(*) Compiler Construction tools :-

1) Partial Generator
2) Scanner Generator
3) Syntax Directed translation engines
4) automatic code generator
5) Data flow engines

(*) The role of lexical Analyzer :-



tokenization :-

lexeme(?) → Tokens

int a,b;

first i ની જોડી પછી જોડી કે tab, newline કે space હોય ત્યાં સુધી તો તે stop કરી દેશે.

So, first lexeme આ int આવશે.

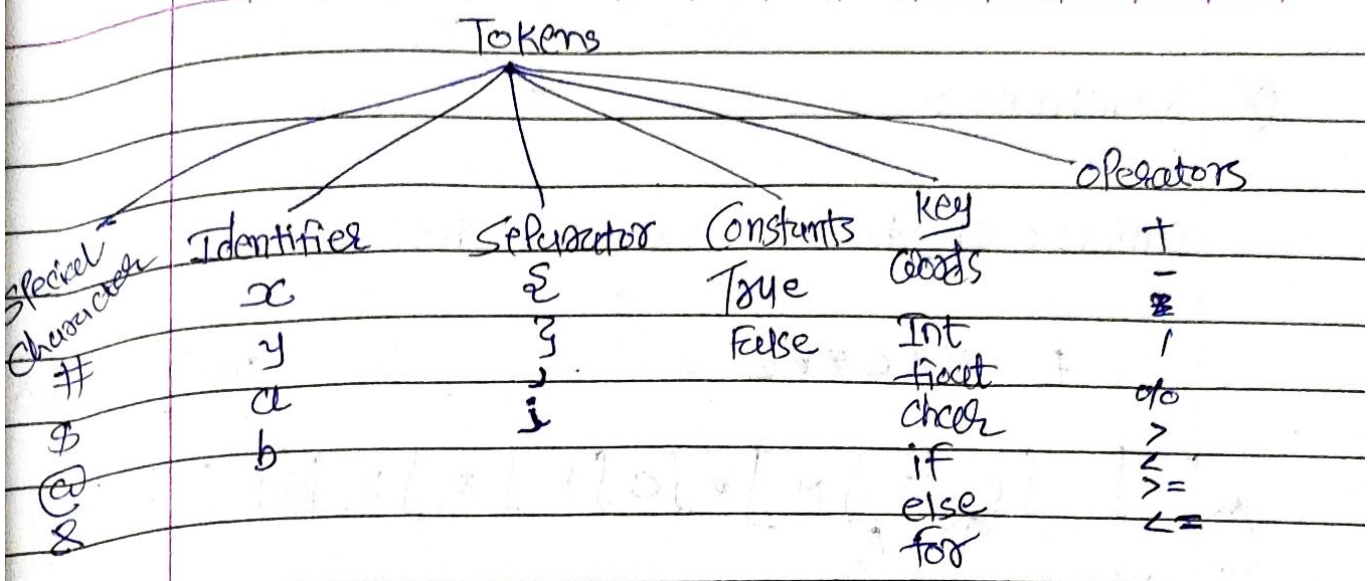
So, int
a
,
b
;

આ બધા lexeme અલગ અલગ પછી એક lexeme નો token બનશે ત્યારે.

↳ eliminate Comments & whitespace(tab, blankspace, newline)

↳ lexeme remove કરવી લાઈન

Date
Page



Example: int main()
{
int a=20, b=30;
if(a<b)
&return(b);
else
&return(a);
}

total token = 32

(*) Print("i=%d, &i=%d", i, &i);

" " વાળી જગ્યા સિવાય બધું જ lexeme કહેવાય.

a**b
↓
consider 1 lexime

(*) Gives Error messages :-

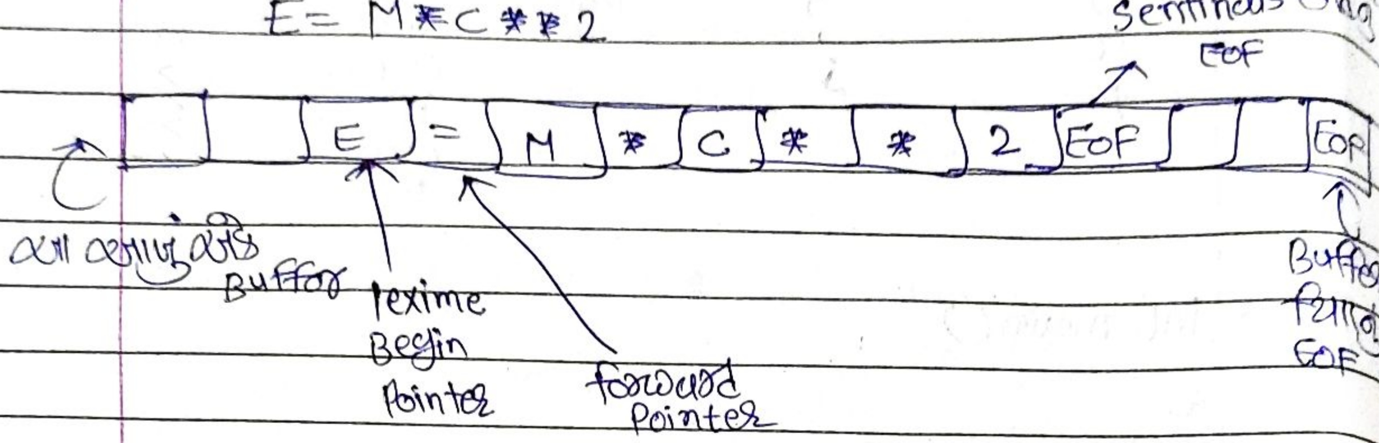
↳ ~~exeding~~ ~~for~~ Exceding the length, unmatched string

↳ ~~brackets~~, ~~Comments~~ illegle Comments,

(*) Sentinals :-

Buffers :- It stores the Data for short amount of time

E= M*C**2



↳ lexime Begin Pointer & forward Pointer વચ્ચે જે Data છે તેને compare કરવી પછી જો તેમના વચ્ચે કોઈ Relation નહિ હોય તો પછી બીજો બીજો move કરાવવા કરવી

↳ જ્યારે જો Relationship હશે તો તે વચ્ચેનો બીજો lexime conside કરવી.

IMP Que:-1 The tokens & Associated attribute values for the statement E = M * C ** 2

(*) Recognization of Tokens :-

(*) Recognization of identifier
Recognization of Delimiters
Recognization of Relational operator
Recognization of keywords
Recognization of number

⇒ letters → a|b|c|---|z
A|B|C|---|Z
Digit → 0|1|2|---|9
id → letter (letter|digit)*

⇒ Delimiters:- newline | tabs | space
DC → (Delimiters)+

Soln of Que:-1

id = identifier

<id, Pointer to symbol table, Entry for E>
<assignment operator, >
<id, Pointer to symbol table, Entry for M>
<Multiplication operator>
<id, Pointer to Symbol table, Entry for C>
<Exponential operator>
<num, constant 2>

Date
Page

Experiment-1

1) Write a Sample Program in C or C++ language and observe Compiling, linking & loading process using gcc Compiler. (find difference in size of files generated.) (find which file generated after each step.)

2) Study and answer the following Questions :-

(1) Discuss the following :-

① Compilers ② Interpreter ③ linkers ④ loaders

(2) GCC :-

① What is gcc ? explain it's features.

② in which language is gcc written ?

③ What Standard optimization are supported by gcc ?

④ Explain following options of gcc. -O -S, -C, -save-temps, -l, -V, -c-static, size <exefilename>, objdump -hst <objfile>

(3) LLVM :-

① understand & Discuss these (following) keywords :- open CL, open GL, open MP, CLANG, LLVM, BISON

② What is LLVM ? explain it's features.

③ Which languages are using LLVM

④ What are the components of LLVM ?

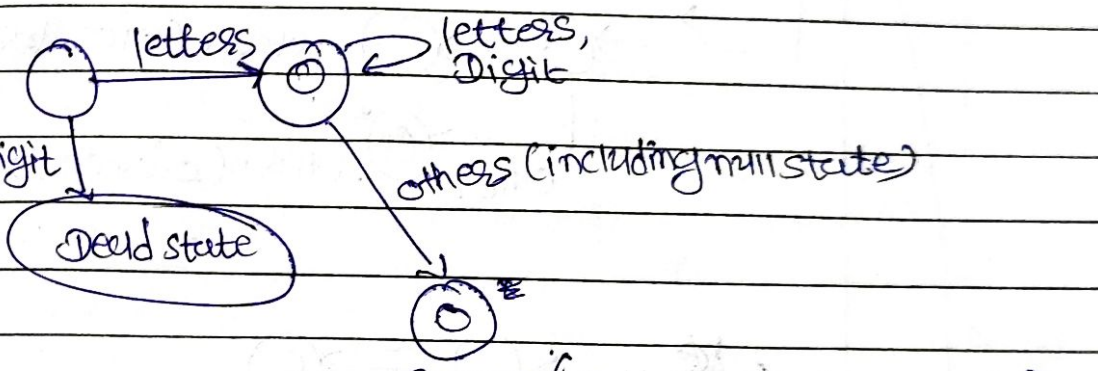
⑤ What is Clang ? What are it's features ?

3) Find & explain features of toy language like that language. (4) Explain how flex can be use to Design programming language?

(5) Give Design of your toy language.

15/12/2022 LT

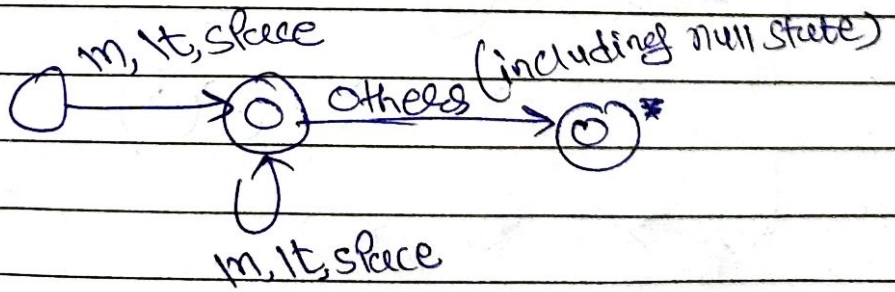
id → letters (letters/digit)*



(*) Recognization of Delimeter :-

Delimeter → newline(\n) | space | tab(\t)

ws → Delimeter+

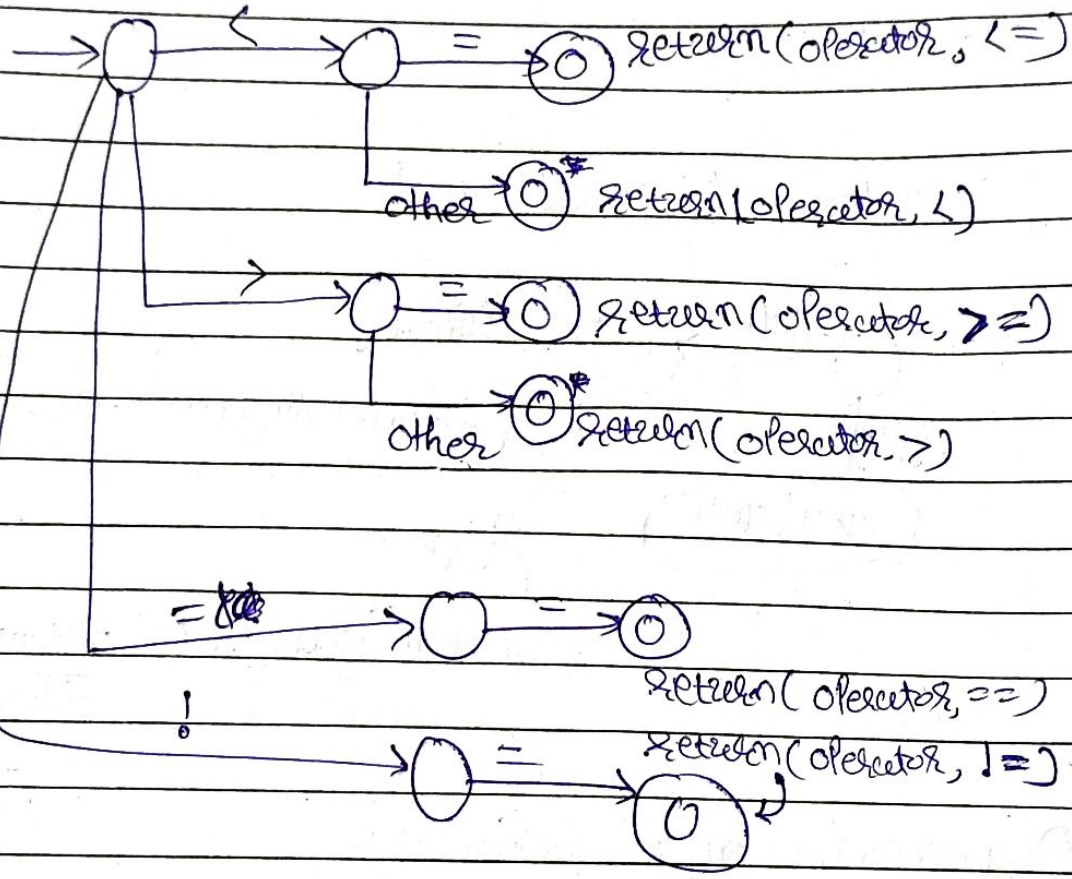


(*) Recognization of Relational Algebra:-

~~(<, >, =, +, -, ÷, *, ----)~~

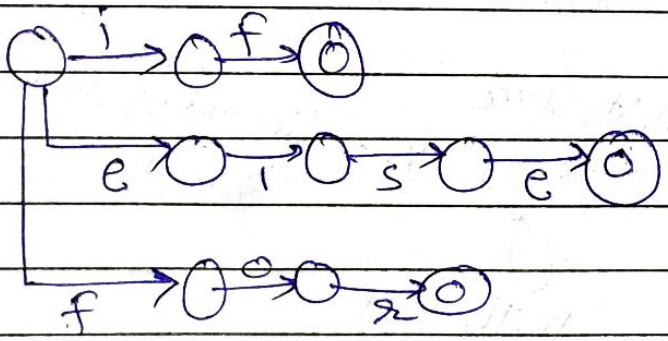
~~operator → <|>|=|+|-|---|*~~

operator → <|>|<=|>=|==|!=



(*) Recognization of keywords:-

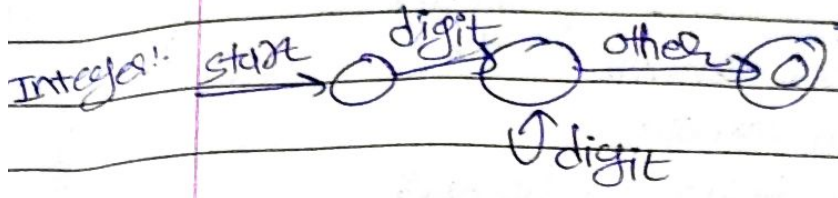
Rekey → if|else|for



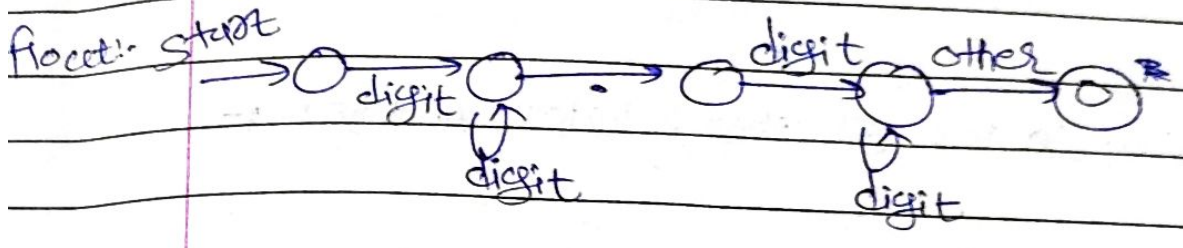
(*) Recognization of Numbers :

digit → 0|1|2|---|9

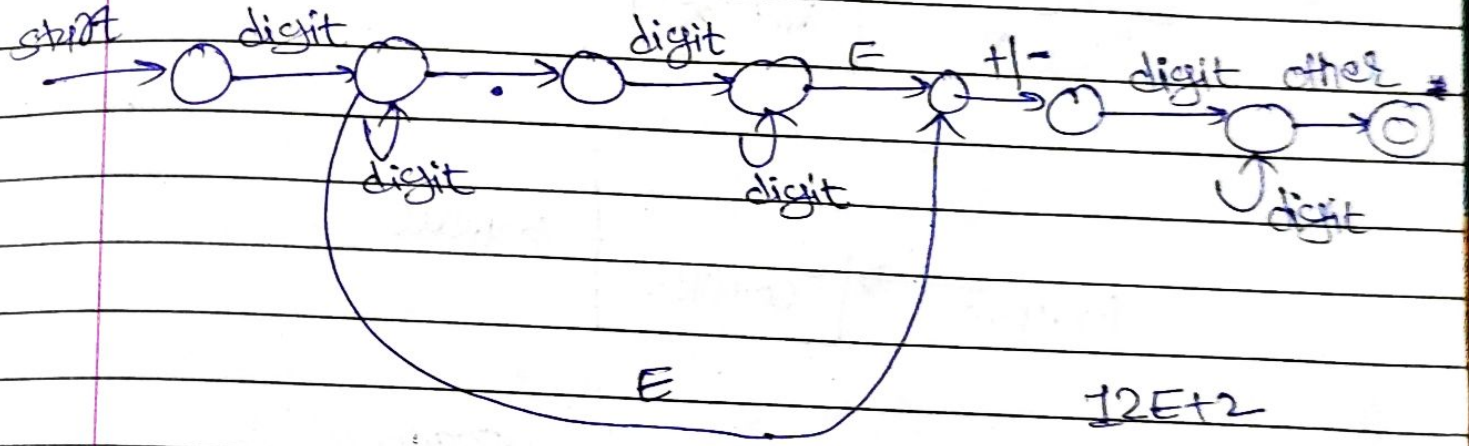
Integer :-



float :-



Exponential :-



12E+2

12.21E+2

Date:- 16/12/2022

(*) num → ~~digit⁺ . (. digit)⁺~~ digit⁺ (.digit⁺) ? (E (+|-) ? digit⁺) ?

Recognization of Number

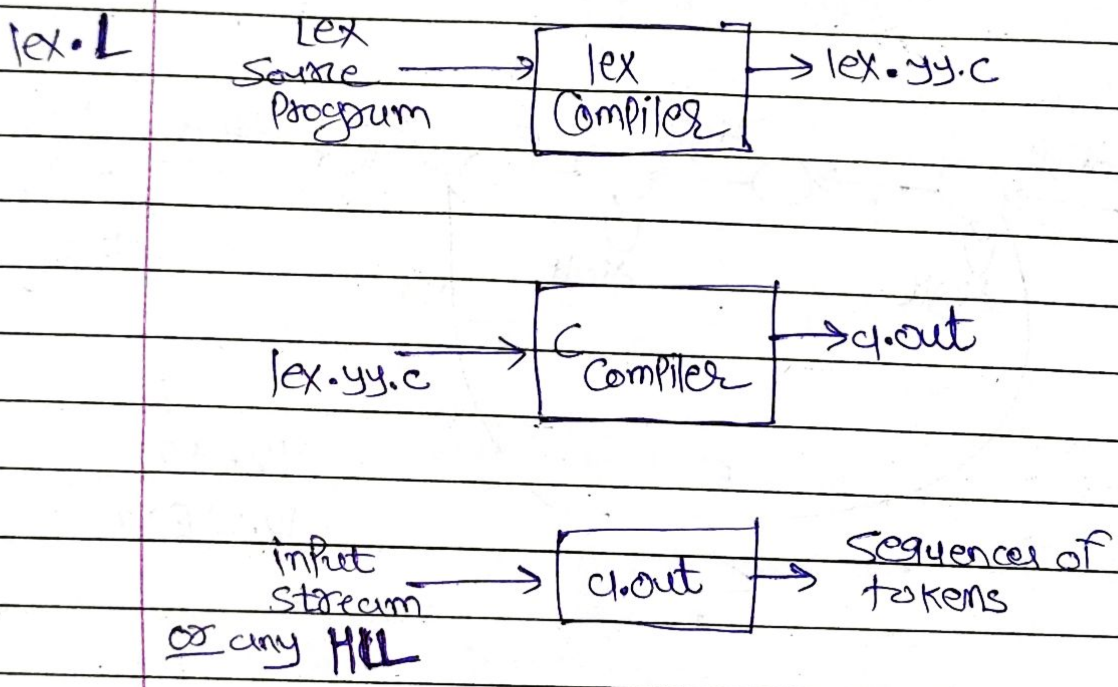
? → represents 0 or 1

→ આ આવી પણ શકે અને ના પણ આવે

O = (.digit)⁺ આ નહિ આવે

12.5E+9
12.5E-5
12

(*) A language for specifying lexical Analyzer :-



(*) Lex Specification

start with a → ^a

end with a → a$

abc	^a	yywrap() → If called by lex or when input is existed, else it will be return 0.
[a-z]	a$	yylex()
[a-z]*	[0-9]+	
[A-Z, a-z]+	yytext	

↓ Pointer to a input string

It reads I/P stream & generate tokens to the regular expressions

Declaration :-

%{

%}

transition rules → Pattern (actions)

%%

auxiliary procedures

Example:-

```
%{
    #include <stdio.h>
    /* ------ */
%}
%%
%%
```

Transition Rules:-

"hii" { printf ("How are you?");}

.* { printf ("Wrong string");}

other

%%

```
main()
{
    printf("Enter I/p");
    yylex();
}
int yywrap()
{
    return 1;
}
```

Date:-20/12/22

Examples: Find lexeme, tokens & Patterns of the following eye:-

1) int x=5;

lexeme	tokens	Patterns
int	Keywords	Keyword
x	Identifier	letter(letter/digit)
=	Operator	Relational operator(<,>,=)
5	Constant	Constant for a number
;	Separator	Separator

Rules of erroring:- ① exceeding the length of identifier
like int a = 2154321254367; → આ int માં store ના થાય so error

② appearence of illegal character
↳ like printf("6th - I"); $
↑ illegal character

③ unmatched stream
→ આ string શરૂ થયા
તો ત્યાંથી end
નથી થતું તો એ
unmatched stream
થઈ જાય.

Expo-2 Find out lexical errors

① `Int x;`
ans:- No lexical errors

② `int x,y;`
ans:- No lexical errors

③ `float m=5..7;`
ans:- lexical error
5..7
↑ Replacing incorrect character

④ `int A=#12;`
↳ lexical error

⑤ `string s="success;`
↳ lexical error

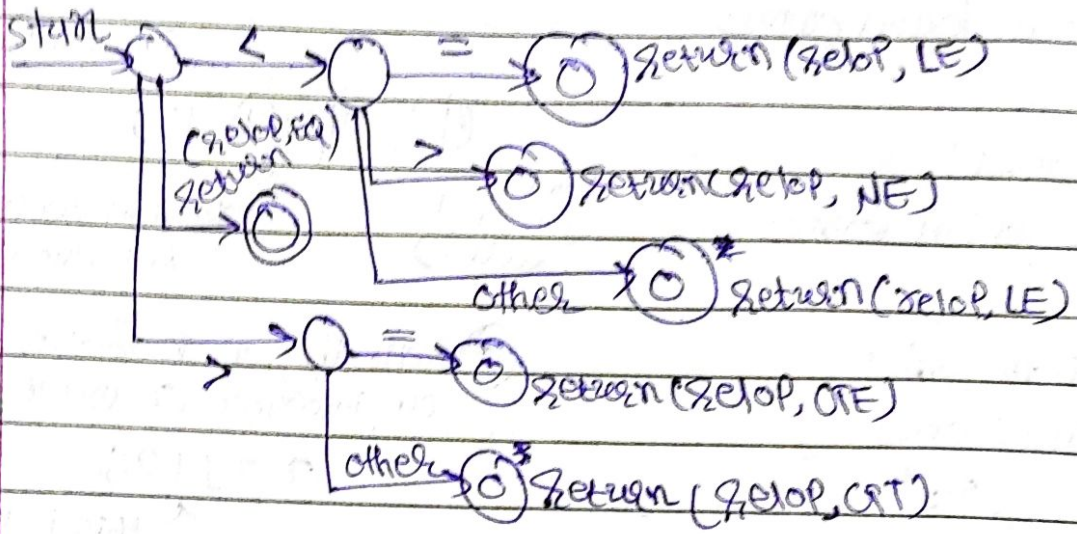
④ `int 6a=75;`
identifier rules not satisfy
↑ starting with number is not Allowed

⑤ Replacing a character with an incorrect character
`x = 1$2;`
↑ this is incorrect character

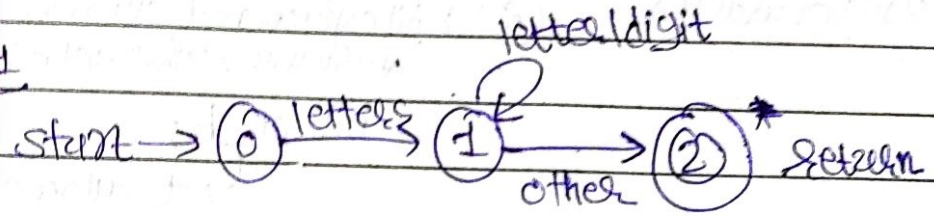
⑥ whitespace not allowed between variable (character)
`int a_b;`
↑ not allowed

⑥
```
#include <stdio.h>
main()
{
    int x=5,y=6;
    char *a;
    a=&x;
    x=56xab;      → Here lexical error
    printf("%d",x);
}
```

(*) Recognition of Relational Operators :-

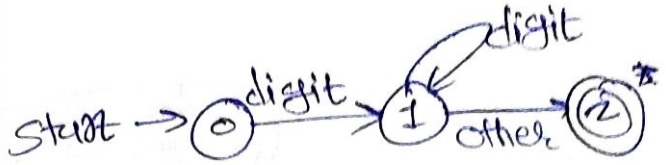


Example:-1




```
int state = 0,
        token nexttoken()
        {
          While(1)
          {
            Switch(state)
            {
              case 0:
              c = nextchar();
            if( isletter(c) )
                state = 1;
            else
                state = fail;
            break;
```

H.CO.



Date
Page

```
case 1 :
    c=nextchar();
    if(isletter(c))
      state=1;
  else if (isdigit(c))
      state=1;
  else
    state=2;
  break;
```

(*)

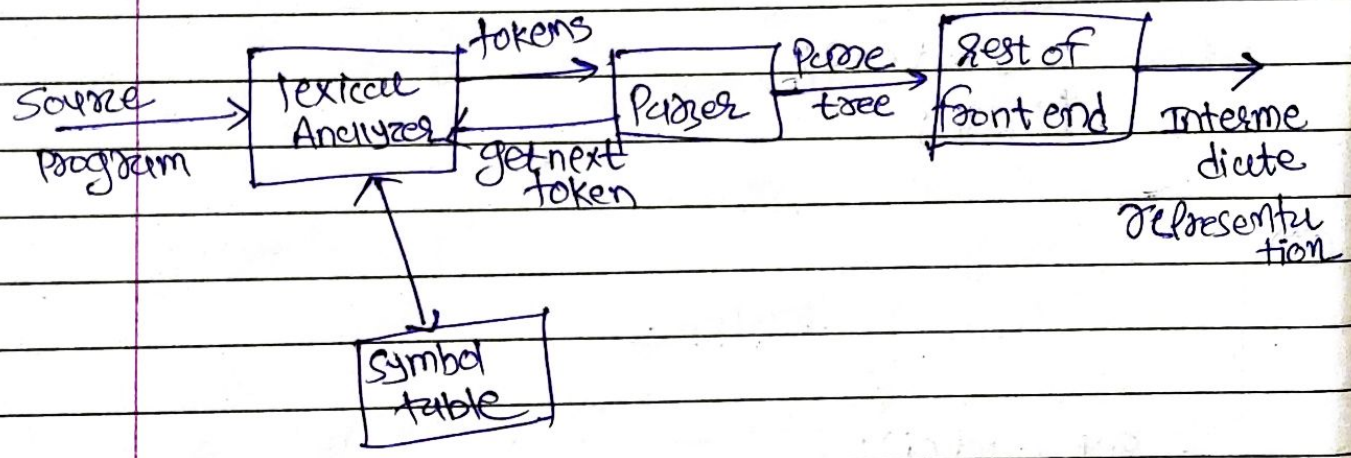
```
set fact(i);
  install-id(), return(gettoken());
```

⇒ install-id is used to access the buffer & it's called to attribute function.

(*) Syntax Analyzer :-

↳ It generates parse tree

↳ The Role of the Parser :-



(*) Context-Free Grammer :-

↳ Context-Free Grammer is a 4 tuple.

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

V = finite set of symbols called as nonterminals or variable (It may be in uppercase, usually S is used for starting symbol, some Italic name are also allowed Example: Expr, stmt)

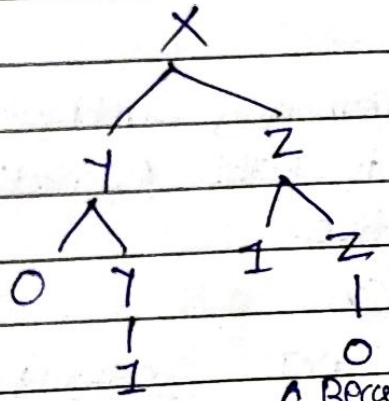
T = set of symbols that are called as terminals (lowercase letters, operator symbol like +, -, *, /, punctuation symbol like (,), $, ;, digit symbol like 0 to 9, bold face string like id, if)

P = Set of Productions ($\alpha \to \beta$)

S = It is a member of V
It is called as Starts

Examples $X \to YZ$
$Y \to 0Y \mid 1$
$Z \to 1Z \mid 0$
ω : 0110

Parse tree:-

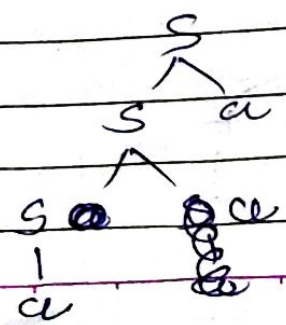


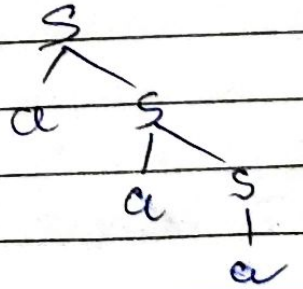
2 ambiguous (Because no other ways to generate 0110)

ambiguity :- A Grammer that produces more than one Parse tree for same sentences it's said to be ambiguous Grammer.

(1) $S \to Sa \mid aS \mid a$ this Grammer is ambiguous

ω : aaaa





Unambiguous ←
A → A$A | T
T → T#F | F
F → F@E | E
E → id

H.W. → $A \to A\$A \mid A\#A \mid A@A \mid d$

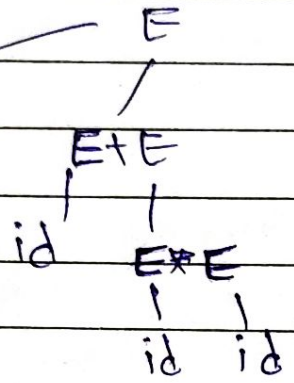
Check if it is ambiguous or not? If yes remove it.

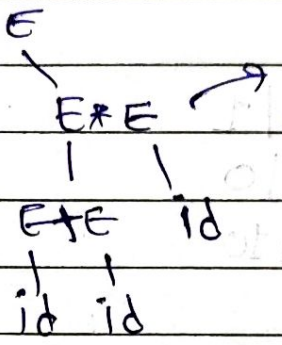
Date
Page

(2) $E \to E+E \mid E*E \mid id$ (this Grammer is ambiguous)

id + id * id

left most Derivation tree





right most Derivation tree

Rules :- for removing ambiguity

① $ and ↑ is Right Associative. Remaining all else left Associative.

② $ < # < @

③ or < and < not (equally priority)

④ − < + < * < / < ↑ = $

⑤ + < * < O*

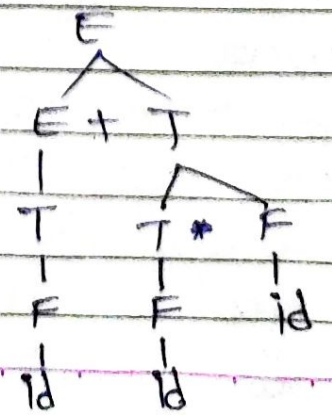
After Removing the ambiguous the Grammer is-

$E \to E+T \mid T$

$T \to T*F \mid F$

$F \to id$

id + id * id



Date
Page

(1) Study flex manual & find the answer for the following questions:-

<1> What is flex

<2> What are characteristics of flex?

<3> What is format of flex?

<4> Explain the steps to use flex?

<5> Is following flex input program valid? If yes How is the result & scanner?

<6> Generate scanner which identifys tokens 'Username' & Display the current login username in response.

<6> Generate scanner using lex which simply counts the no. of lines & no. of character in input & Display it's when endof file encountered in inputs

Ans-3.6 Toy language features:-

(1) Every toy language uses entry & exit Points

hi bhai } use entry & exit Points
bye bhai } in bhai language representations

(2) Variables:-

bhai ye hai keyword is used for variables in bhai language.

(3) Datatypes:-

numbers & string are just like any other language all values are indicated with needle. Sahi and

Date
Page

Cout represents boolean values.

(4) Outlet:-

Any console outlet can be printed with bas bhai.

(5) Control structure:- Break the loop with bas kar bhai (break) and continue within it with agla dekh (continue) bhai.

(6) errors:- It throws a syntax error with kya kar raha hai tu statements.

Ans-1 Flex (fast lexical analyzer generator) is a free & open-source software alternative to lex. It is a computer program that generates lexical analyzers.

Ans-3 the flex input file consists of three sections, separated by a line with just '%%' in it: definitions %% rules %% user code. The definitions section contains declarations of simple name definitions to simplify the scanner specification, & declaration of start symbol.

Ans-4 Step:-1 An input file describes the lexical analyzer to be generated named lex.l is written in lex language.
Step:-2 then Compiler compile lex.yy.c file into an executable file called a.out.

Date:- 22/12/2022

Que:-1 bExp → bExp or bExp | bExp and bExp |
NOT bExp | True | False

Soln:- This is ambiguous.

bExp → bExp or T | T
T → T and F | F
F → NOT F | K
K → True | False.

Que-2 R → R+R | RR | R* | a | b | c } that is ambiguous Grammer
E → E+T | T
T → TF | F
F → F* | a | b | c } this is not an ambiguous Grammer

Soln:- R → R+M | M
M → Ms | S
S → s* | a | b | c
E → E+T | T
T → TF | F
F → F* | a | b | c } an ambiguous and solution as it is left recursive.

$A \to A\alpha \mid \beta \mid c$

After removing left Recursion → $A \to \beta A' \mid cA'$

$A' \to \alpha A' \mid \varepsilon$

Date
Page

(*) left Recursion :-

$A \to A\alpha \mid \beta$

this Problem is left Recursion

Algorithm :-

Input :- CFG

output :- Equivalent Grammer with No Recursion

Method :-

(1) If we have left Recursive tree production

$A \to A\alpha \mid \beta$

(2) where β does not begin with A. then we can eliminate left Recursion by replacing this Pair of Production with $A \to \beta A'$

$A' \to \alpha A' \mid \varepsilon$

⇒ There are two types of left Recursion :-

(1) Direct left Recursion

(2) InDirect left Recursion

(1) Direct left Recursion :-

1) $A \to ABd \mid Aa \mid a$

$B \to Be \mid b$

Sol :- No ambiguity are there.

$A \to aA'$

$A' \to BdA' \mid aA' \mid \varepsilon$

$B \to bB'$

$B' \to eB' \mid \varepsilon$

Date
Page

2) $A \to A\alpha B \mid \gamma$
$B \to B\beta b \mid \gamma\beta$
$C \to Cc \mid \varepsilon$

Soln:- $A \to \gamma A'$
$A' \to \alpha B A' \mid \varepsilon$
$B \to \gamma B'$
$B' \to \beta b B' \mid \varepsilon$
$C \to \varepsilon C'$
$C' \to cC' \mid \varepsilon$
} same as $C \to Cc \mid \varepsilon$

(2) Indirect left Recursion :-

1) $S \to Aa \mid b$
$A \to Ac \mid Sd \mid \varepsilon$
$A \to Ac \mid Aad \mid bd \mid \varepsilon$ ($S \to Aa \mid b$ ની use કરી)

Soln:- $A \to Ac \mid Sd \mid \varepsilon$, $S \to Aa \mid b$

$A \to bdA'$
$A' \to cA' \mid \varepsilon$
$A \to bdA'$
$A' \to adA' \mid \varepsilon$
} same like this $A \to bdA'$, $A' \to cA' \mid \varepsilon$ } - (1)

$A \to \varepsilon A'$
$A' \to cA' \mid \varepsilon$
$A \to \varepsilon A'$
$A' \to adA' \mid \varepsilon$
} same like this $A \to A'$, $A' \to cA' \mid adA' \mid \varepsilon$ } - (2)

Date ______
Page ______

Note:- Ambiguity માટે હમેશાં first & last Nonterminal Same હોવું જોઈએ.

$E \to E+E \mid E*E \mid id$

first & last same — first & last same So It is ambiguous

① & ② નુ final answer:-

$A \to Ac \mid Sd \mid \varepsilon$

$S \to Aa \mid b$

$A \to bdA' \mid A'$

$A' \to cA' \mid \varepsilon \mid adA'$

⊛ left factory:-

$A \to aAb \mid aAc \mid aAd$ → that is called left factory

Algorithm:-

Input:- will take Grammer G

output:- equivalent non left factory Grammer

Method:- for each non-terminal A find the prefix α common two or more of it's alternative. Place one of the A production.

$A \to \alpha\beta_1 \mid \alpha\beta_2 \mid \alpha\beta_3 \mid \alpha\beta_4 \mid \gamma$ → ક્યાંય કે જે α થી start ના હોવું જોઈએ

Where γ is present all alternative that do not begin with α.

$A \to \alpha A' \mid \gamma$

$A' \to \beta_1 \mid \beta_2 \mid \beta_3 \mid \beta_4$

→ Repeatedly apply ~~that~~ there transformation until these two alternative's for a Common Prefix.

Date ______
Page ______

$A \to abc \mid abd \mid acd \mid ad \mid d$

$bc \mid bd \mid cd \mid d$

Example:-1 $S \to iEtS \mid iEtSeS \mid a$ ← આ small s છે બાકીના capital છે.

$E \to b$

Find If there is ambiguous or not? left Recursion is there or not? then apply left factory.

Sol^n:- left factory:-

$S \to iEtSS' \mid a$

$S' \to \varepsilon \mid eS$

$E \to b$

Example-2 $A \to aAb \mid aA \mid a$ → આને a તરીકે નહિ લેવાય કારણ કે a છે તે ર... start માંથી કરેલી.

Sol^n:- left factory:-

~~$A \to aAA' \mid a$~~

~~$A' \to b \mid \varepsilon$~~

$A \to aA'$

$A' \to Ab \mid A \mid \varepsilon$

$A' \to AA'' \mid \varepsilon$

$A'' \to b \mid \varepsilon$

Example:-3 $A \to ad \mid a \mid ab \mid abc \mid b$

Sol^n:- left factory:-

$A \to aA' \mid b$

$A' \to d \mid \varepsilon \mid b \mid bc$

$A' \to bB' \mid d \mid \varepsilon$

$B' \to c \mid \varepsilon$

Que માં first ambiguity જોવાની જો તે હોય તો remove કરવાની પછી જે ans મળે તેમાં left Recursion કે left factory જે લાગુ પડે તે લગાવવું. જો બંને પડતું હોય તો first left Recursion લગાવવું પછી જે ans આવે તેમાં જોવાનું left factory છે કે નહિ એમ.

Example:-4 $A \rightarrow XBYA \mid XBYAZA \mid a$

$B \rightarrow b$

Soln:- $A \rightarrow XBYAA' \mid a$

$A' \rightarrow ZA \mid \varepsilon$

$B \rightarrow b$

Example:-5 $E \rightarrow E+E \mid E*E \mid id$

↳ this is ambiguous

Soln:- after removing ambiguity:-

$E \rightarrow E+T \mid T$

$T \rightarrow T*F \mid F$

$F \rightarrow id$

To Eliminate left Recursion:-

Rule for left Recursion:

$A \rightarrow A\alpha \mid \beta$

$A \rightarrow \beta A'$

$A' \rightarrow \alpha A' \mid \varepsilon$

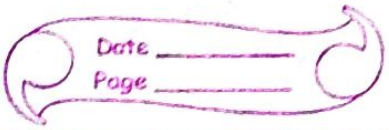
$E \rightarrow TE'$

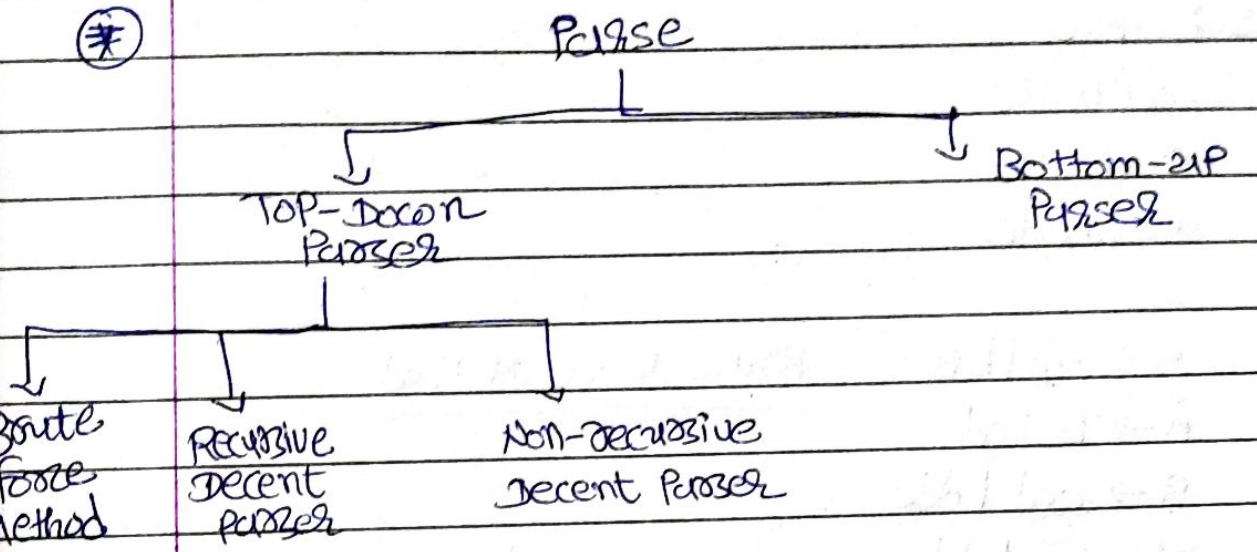
$E' \rightarrow +TE' \mid \varepsilon$

$T \rightarrow FT'$

$T' \rightarrow *F \mid \varepsilon$

$F \rightarrow id$

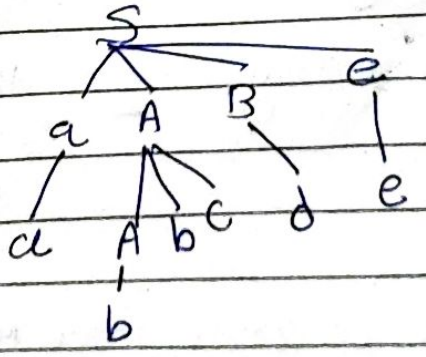




Top-Down Parser :-

↳ Generate Parse tree for the given input string with the help of grammer production by expanding the non-terminals. in this It will start from start symbol and ends on the terminals. It uses left most derivation.

S → aABe
A → Abc | b
B → d
string :- abbcde

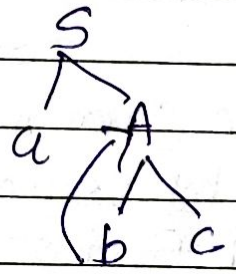


↳ at every point we have to decide what is the next production we should use.

$S \rightarrow aABe$
$\rightarrow aAbcBe$
$\rightarrow abbcBe$
$\rightarrow abbcde$

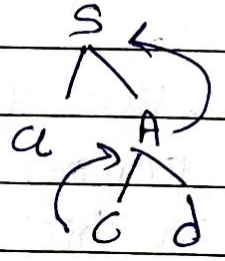
Example:- $S \rightarrow aA \mid bB$ Brute force Method
$A \rightarrow bc \mid cd$
$B \rightarrow ccd \mid ddc$
String:- bddc

Soln:- (I)



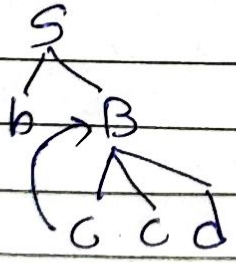
Not a original string

(II)



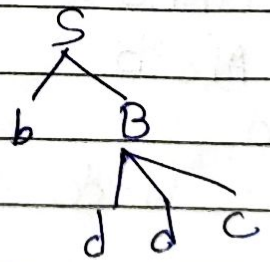
Not a original string

(III)



Not a original string

(IV)



this is a original string

✓

time Complexity of Brute force Method:-
$O(2^n)$

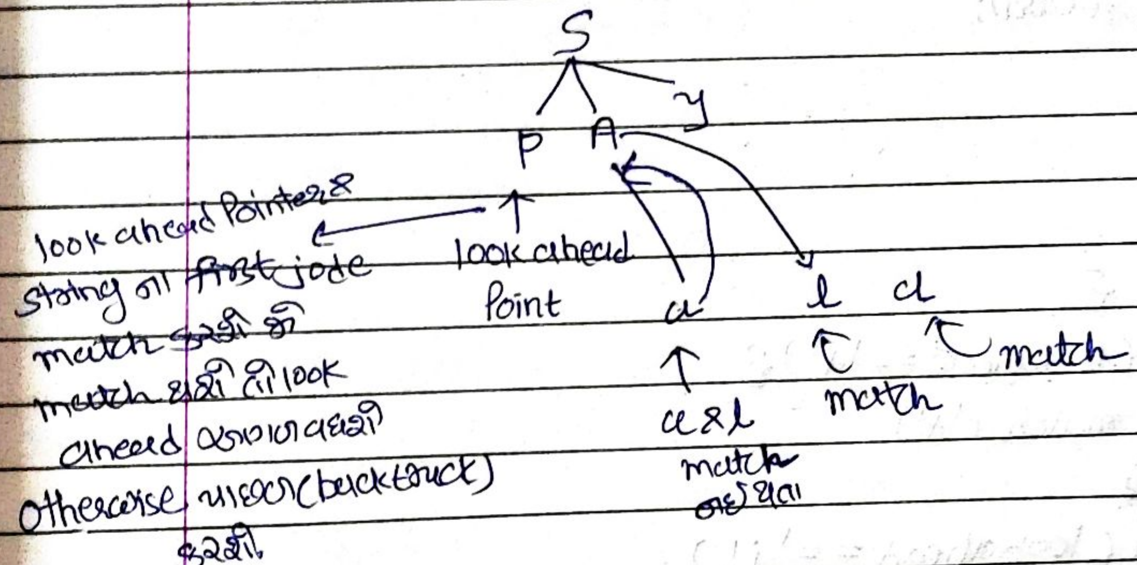
first ambiguous કરવાનું ambiguous ના હોવું જોઈએ left Recursion ના હોવું જોઈએ & left factory ના હોવું જોઈએ તો જ પછી Brute force, Recursive Decent ની કોઈ પણ Method વાપરવી.

(*) Recursive Decent Parser :-

Example:-1 S → pAy

A → a | la

String :- Play
↑



(*) Recursive Decent Parse :-

E → iE'

E' → +iE' | ε

(1) E( )

(2) E'( )

(3) match (token t)

{ S → iEtS | iEtSeS | a
E → b
this is ambiguous Grammer }

```
E( )
{
   if(lookahead == 'i')
   {
      match('i');
      E'();
   }
```

```
if(lookahead=='g')
{
    Print("Sucess");
}
else
{
    return;
}
}

E'() {
if(lookahead=='+'){
    match('+')
if(lookahead=='i')
{
    match('i');
    E';
}
}
else
{
    return;
}
}
```

H.W.

Example:- $S \rightarrow ABc$ Using Recursive Decent Parser

$A \rightarrow 0A1 | \varepsilon$

$B \rightarrow 1B1 | \varepsilon$

$C \rightarrow 1C0 | \varepsilon$