CD

COMPILER DESIGN

* Assignment - 1 *

En. no. 170220107030

Que-1) Explain tokens, lexemes, pattern with ex.

Ans. ① Tokens :-

→ sequence of characters having a collective meaning is known as tokens.

→ ex. total = sum + 12.5

tokens are total (identifier 1) = (relation)
sum (identifier 2) + (operator)
12.5 (constant)

② Lexems :-

→ The sequence of character in a source prog matched with a pattern for a token is called lexems.

→ ex. total = sum + 12.5
lexems are : total , = , sum , + , 12.5

③ pattern :-

→ the set of rules called pattern associated with a token

1702201070 30

Page No.:
Date: / /

Que-2) Explain the Analysis & synthesis model of compilation list the factors that affect the design of compiler. also list major fun. done by compiler.

Ans. There are mainly two parts of process

① Analysis phase :

→ The main objective of the Analysis phase is to break the source code into parts & then arranges these pieces into a meaningful structure.

→ Analysis part is divided into three sub parts

ⓐ Lexical Analysis :- It is also called linear Analysis or Scanning.

⇒ Lexical Analyzer reads the source prog. And then it is broken into stream of units. Such units are called token

⇒ for ex. a = a + b * c * 2

a (id1), = (Assignment sign)
a (id1), + (OP1), b (id2), * (OP2)
c (id3), * (OP2), c (id3)

170210107030

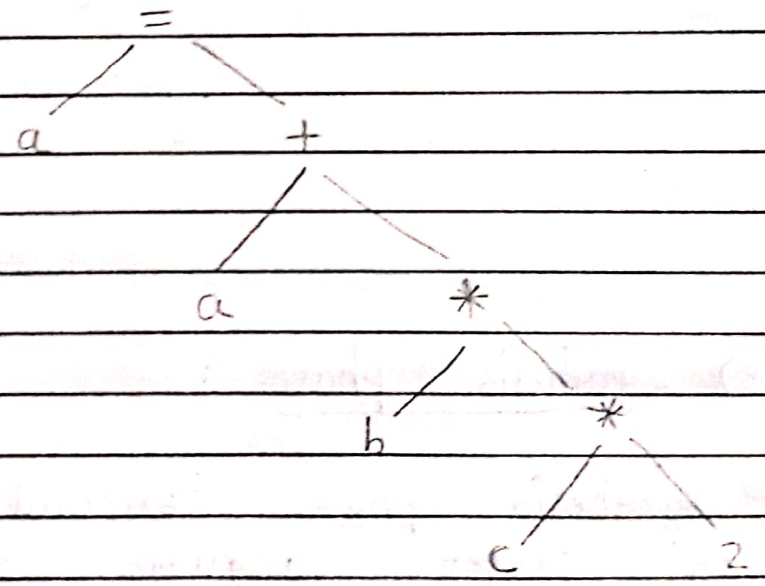
Page No.:
Date: / /

(B) Syntax analysis :

⇒ Syntax analysis is also called hierchical analysis or parsing.

⇒ The Syntax Analyzer checks each line of code & spots every time mistake that the programmer has committed while typing the code.

⇒ If code is error free then syntax Analyzer generates the tree.

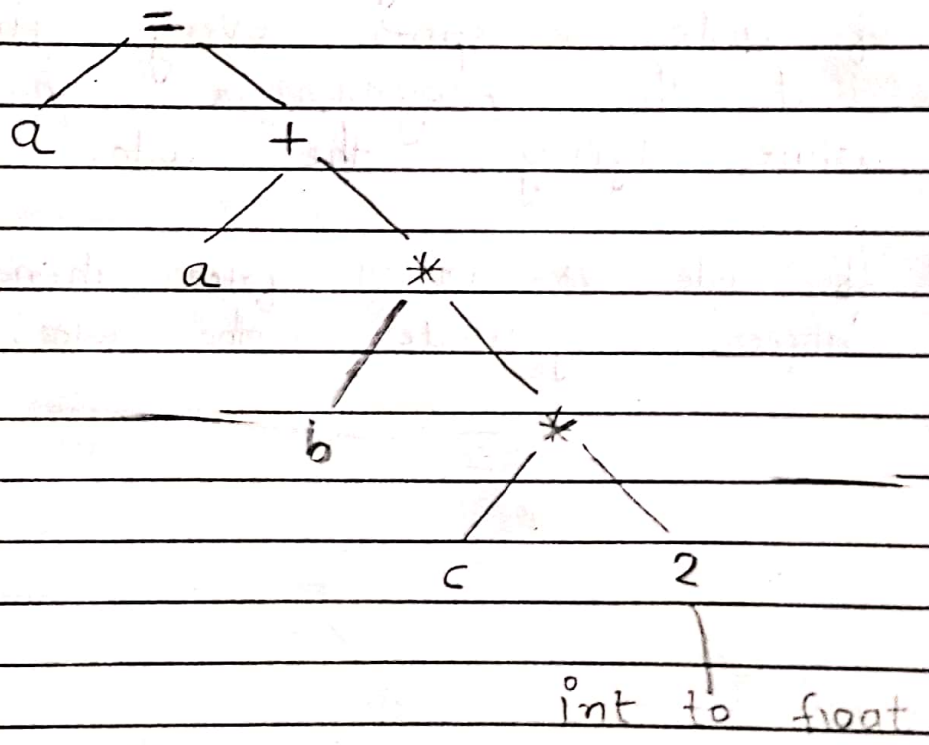


(C) Semantic analysis :-

⇒ Semantic analyzer determines meaning of a source string.

Page No.:
Date: / /

⇒ for example matching of parathensis in the expression or matching of if.... else statement or performing arithmetic operation that are type compatible or checking the scope of operation.



2) Synthesis phase :

⇒ synthesis phase concered with generation of target language statement which has the same meaning as the source statement.

⇒ synthesis part is divided into sub parts.
(A) code optimization
(B) code generation.

170220107030

→ Intermediate code generation is performing betⁿ Analysis & synthesis phase.

Intermediate code generation :-

→ easy It should be easy to produce and easy to translate into target Program.
→ It is also called three address code
→ Three address code consist of a sequence of instruction, each of which has atmost three operands.

```
t1 = int to real (2)
t2 = id3 * t1
t3 = t2 * id2
t4 = t3 + id1
id1 = t4
```

(A) Code optimization :-

→ This is necessary to have a faster execut-ing code or less consumption of memory.

```
t1 = id3 * 2.0
t2 = id2 * t1
id1 = id1 + t2
```

(B) code generation

→ The Intermediate code instructions are translated into sequence of machine instruc

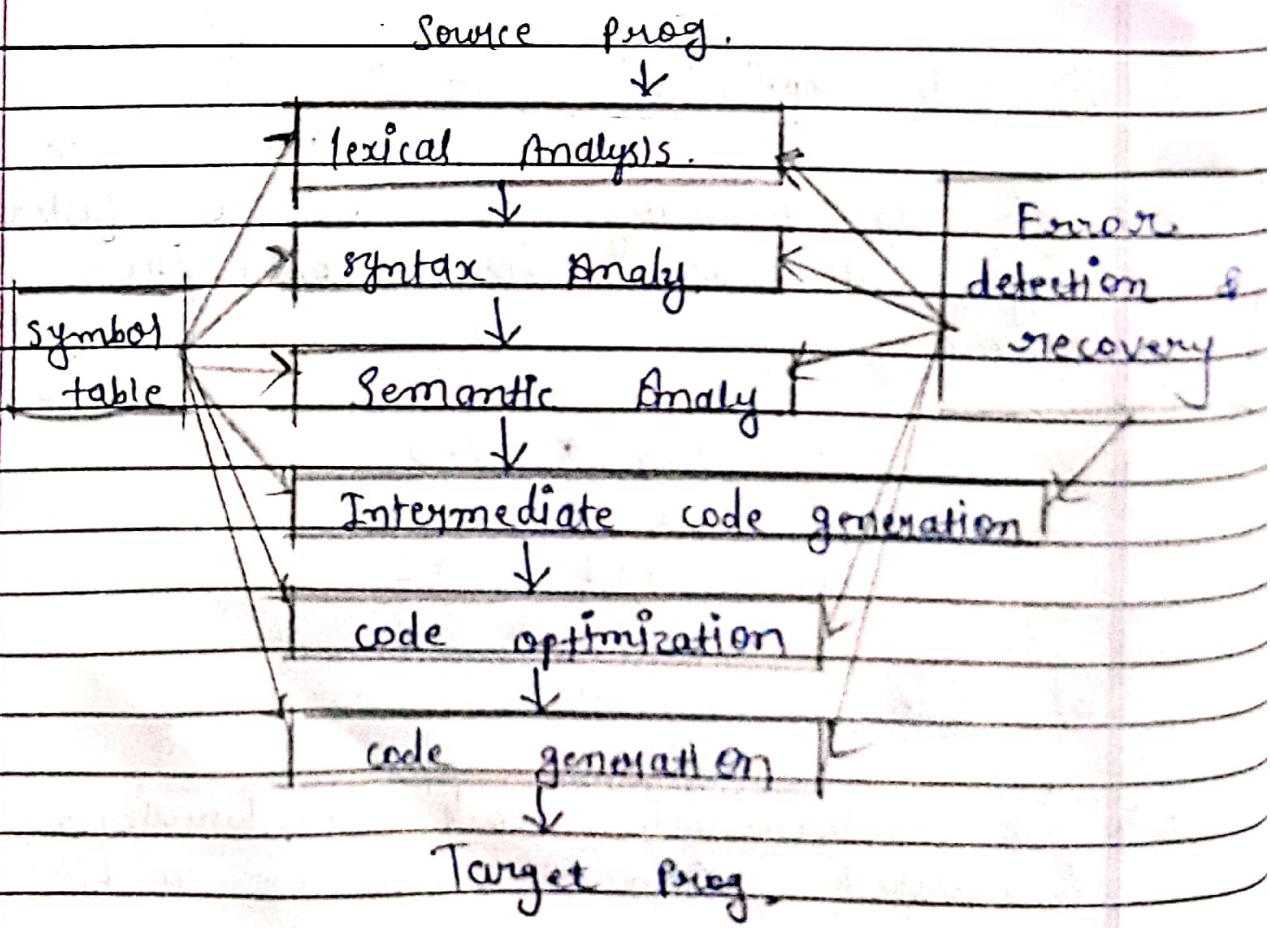
170220107030

```
MOV   id3 , R1
MUL   #2.0 , R1
MOV   id2 , R2
MUL   R2 , R1
MOV   id1 , R2
ADD   R2 , R1
MOV   R1 , id1
```

* Symbol table

→ A symbol table is a table structure used by a lang translator such as a compiler or interepreter.

eg: identifier, fun, keyword, constant, class name, label name



Page No.:
Date: / /

* factor that affect design of compiler.

① Input to the code generator
② target program
③ memory management
④ Instruction selection
⑤ Register allocation
⑥ Evaluation order.

* function of compiler :-

→ Compiler is used to convert one form of program to another.
→ compiler should preserve the meaning of source code.
→ compiler should report errors that occur during compilation process. and compilation must be done efficiently.

Que-3) Write a regular defn for
ⓐ The lang. of all strings that do not end with 01.

Ans. $(0+1)^*\ (00+10+11)$

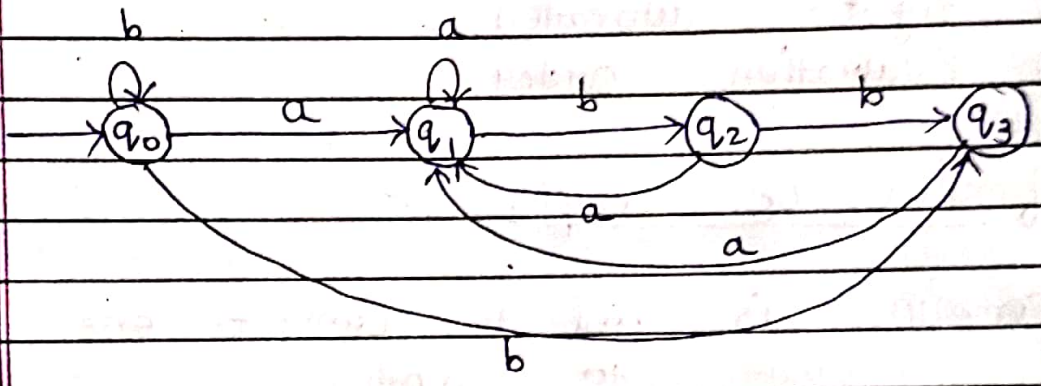
ⓑ All strings of digit that contain no leading 0's.

Ans. $1^* + (1^*\,0\,1^*\,0)^*\,1^*$

Page No.:
Date: / /

Que-4) Construct a DFA for a given regular expression

(a / b)* abb



Que-5) Construct DFA without constructing NFA for following regular expression

a* b* a (a/b) b* a #

n	follow POS (n)
1	{1, 2, 3}
2	{2, 3}
3	{4, 5, 6, 7}
4	{4, 5, 6, 7}
5	{4, 5, 6, 7}
6	{6, 7}
7	{8}

root = {1, 2, 3} → A

construct DFA.

170220107030

Page No.:
Date: / /

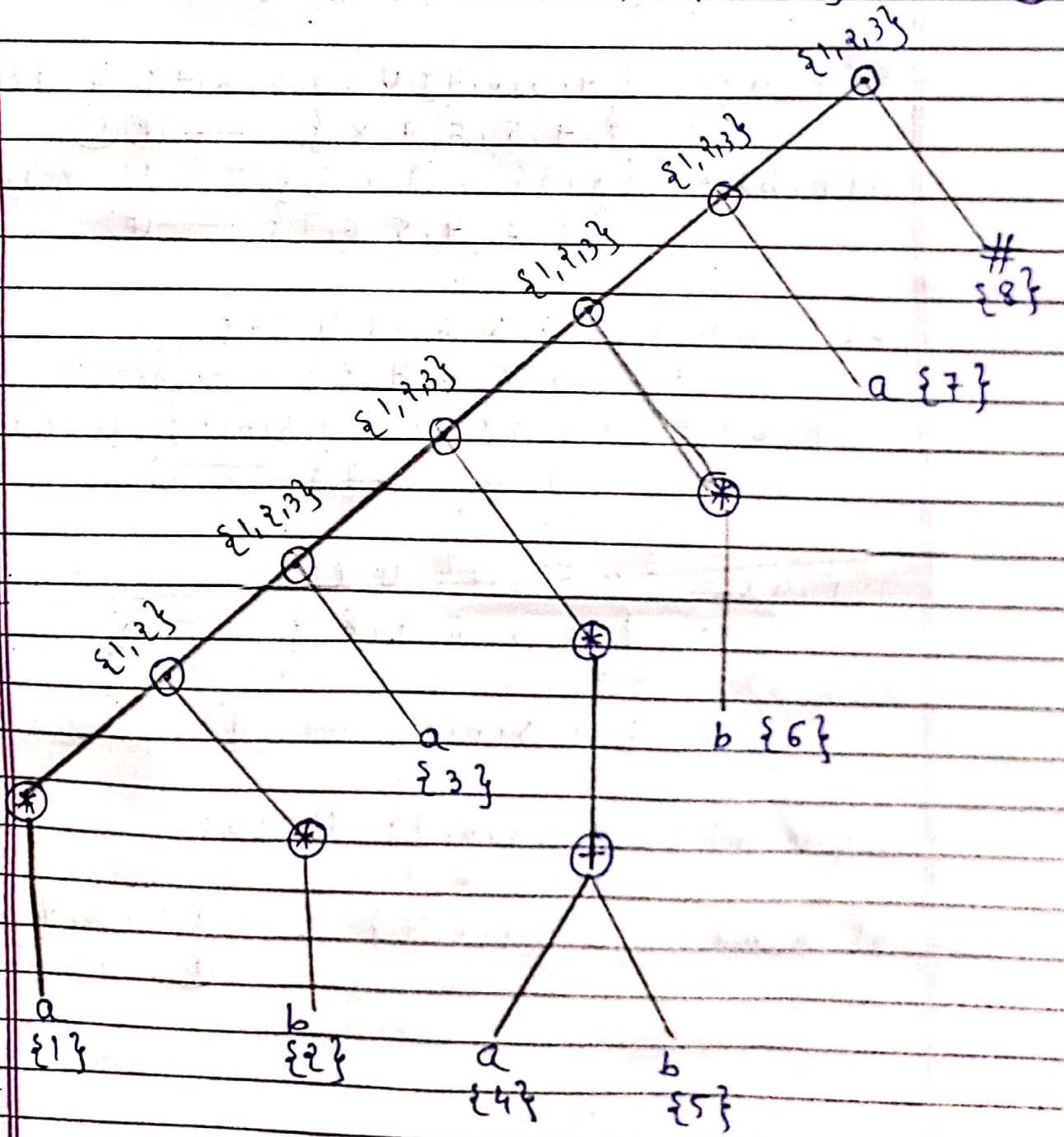
$\delta(A, a) = \{1, 2, 3\} \cup \{4, 5, 6, 7\}$

$= \{1, 2, 3, 4, 5, 6, 7\}$ —— (B)

$\delta(A, b) = \{2, 3\}$ —— (C)

$\delta(B, a) = \{1, 2, 3\} \cup \{4, 5, 6, 7\} \cup \{8\} \cup \{4, 5, 6, 7\}$

$= \{1, 2, 3, 4, 5, 6, 7, 8\}$ —— (D)



170220107030

Page No.:
Date: / /

* $\delta(B,b) = \{2,3\} \cup \{4,5,6,7\} \cup \{4,5,6,7\}$
$= \{2,3,4,5,6,7\}$ —— (E)

$\delta(C,a) = \{4,5,6,7\}$ —— (F)
$\delta(C,b) = \{2,3\}$ —— (C)

$\delta(D,a) = \{1,2,3,4,5,6,7,8\}$ —— (D)
$\delta(D,b) = \{2,3\} \cup \{4,5,6,7\} \cup \{4,5,6,7\}$
$= \{2,3,4,5,6,7\}$ — (E)

$\delta(E,a) = \{4,5,6,7\} \cup \{4,5,6,7\} \cup \{8\}$
$= \{4,5,6,7,8\}$ —— (F)(G)
$\delta(E,b) = \{2,3\} \cup \{4,5,6,7\} \cup \{4,5,6,7\}$
$= \{2,3,4,5,6,7\}$ —— (E)

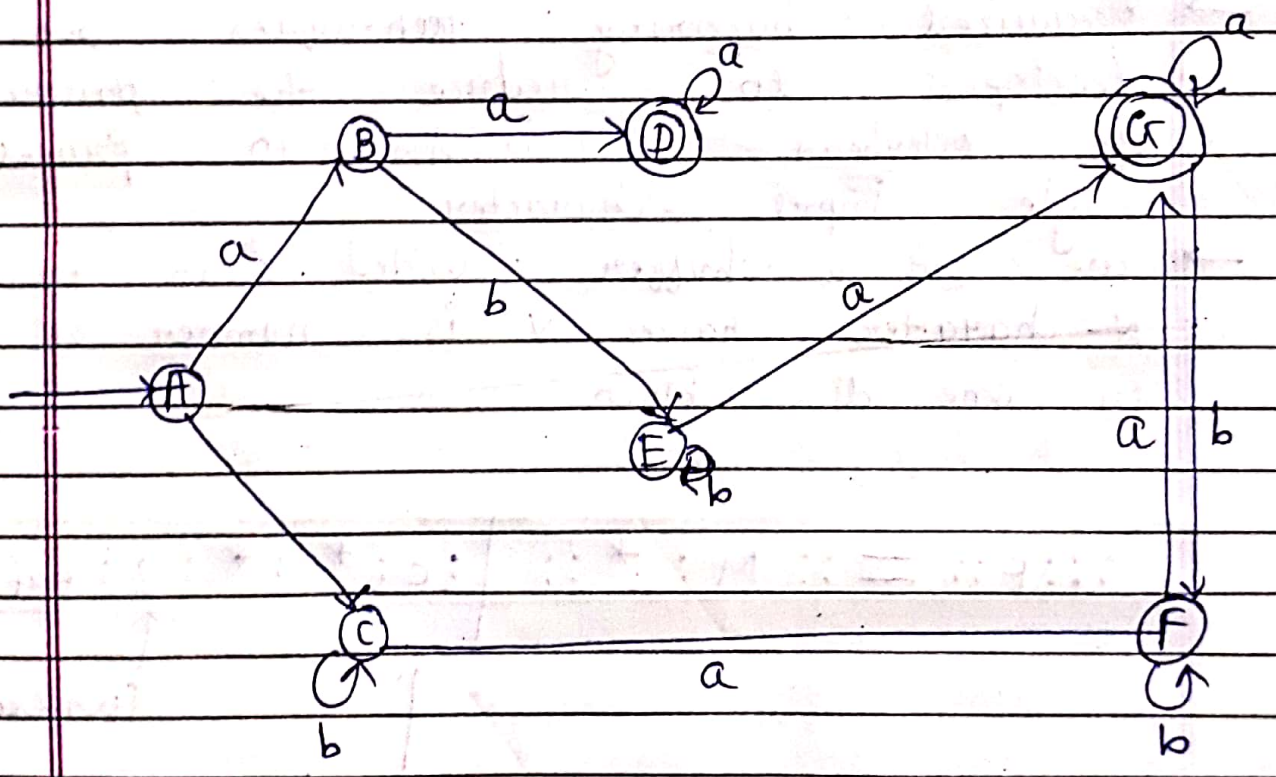
$\delta(F,a) = \{4,5,6,7\} \cup \{8\}$
$= \{4,5,6,7,8\}$ — (G)
$\delta(F,b) = \{2,3\} \cup \{4,5,6,7\} \cup \{4,5,6,7\}$
$= \{2,3,4,5,6,7\}$ —— (E)

$\delta(G,a) = \{4,5,6,7\} \cup \{8\}$
$= \{4,5,6,7,8\}$ — (G)
$\delta(G,b) = \{4,5,6,7\} \cup \{4,5,6,7\}$
$= \{4,5,6,7\}$ —— (F)

170220107030

Page No.:
Date: / /

State	a	b
A	B	C
B	D	E
C	F	C
D	D	E
E	G	E
F	G	F
G	G	F



1702220107030

Page No.:
Date: / /

Que-6) Write a brief note on input buffering techniques.

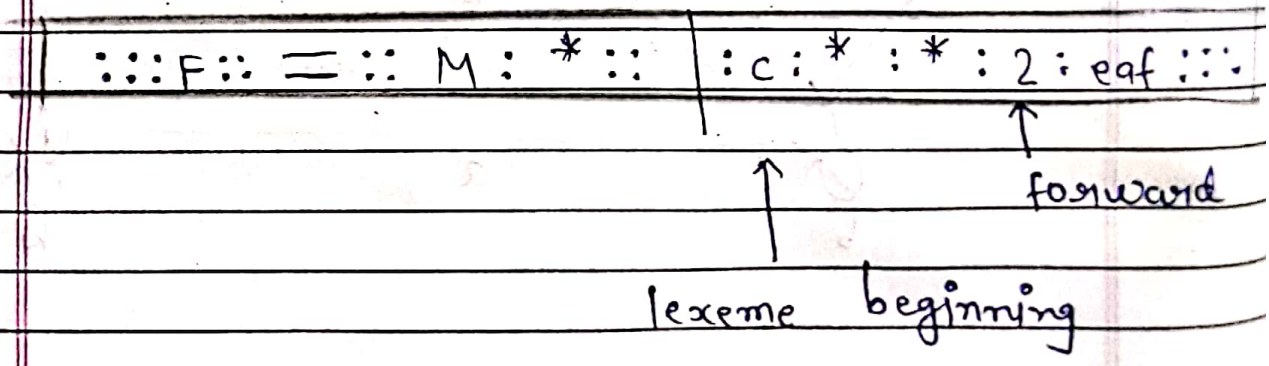
Ans. There Are mainly two techniques for input bulffering.

① Bukker Pair :

→ The Lexical Analysis scans the inputs string from left to right one charact at a time.

→ specialized buffering techniques have been developed to reduce the Amount of overhead required to process single input character.

→ we use a bukker divided into two N- character haves. N is number of char on one disk block



→ we read N input character into each halfs of the buffer :

→ two pointers to the input are maintai -ning & string betn two pointers is the current lexemes.

170220107030

Page No.:
Date: / /

→ if forward pointer is at the end of second buffer half then first is filled with N input chars.

→ if forward at end of first half then
begin reload second half;
forward := forward + 1;

end

else if forward at end of second half then begin
reload first half;
move forward to beginning of first half;

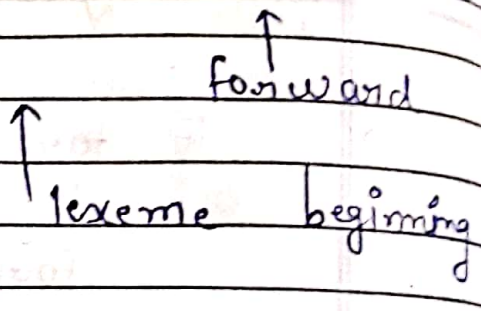
end
else forward := forward + 1;

② Sentinels :-

→ If we use the scheme of buffer pairs we must check each time we move the forward pointer that we have not moved off one of the buffers; if we do, then we must reload the other buffer. Thus for each character read, we make two tests.

170120107030

::: E :: = :: M : * : eof	: C* : * : 2 : eof :: eof



→ forward := forward + 1 ;
if forward = eof then login
if forward at end of first half then begin
reload second half ;
forward := forward + 1 ;

end

else if forward at the second half then begin

reload first half ;
move forward to beginning of first half

end

else terminate lexical Analysis

End ;

170220107036

Page No.:
Date: / /

Que-7) Differentiate Compilers & interepreters

Compiler	Interpreter
→ Scans the entire prog & translates it as a whole into machine code.	It translates programs one statement at a time.
→ It generate Intermedi -te code.	does not generate intermediate code.
→ memory requirement is more.	memory requirement is less.
→ An error is display -ed after entire prog. is checked	An error is displayed for every Instruction interpreted if Any.
→ ex. C compiler	ex. python, ruby

Que-8) Give regular defⁿ for signed & unsigned numbers.

Ans) ⊛ signed number :

→ It contain sign flag, this representation distinguish positive & negative number.

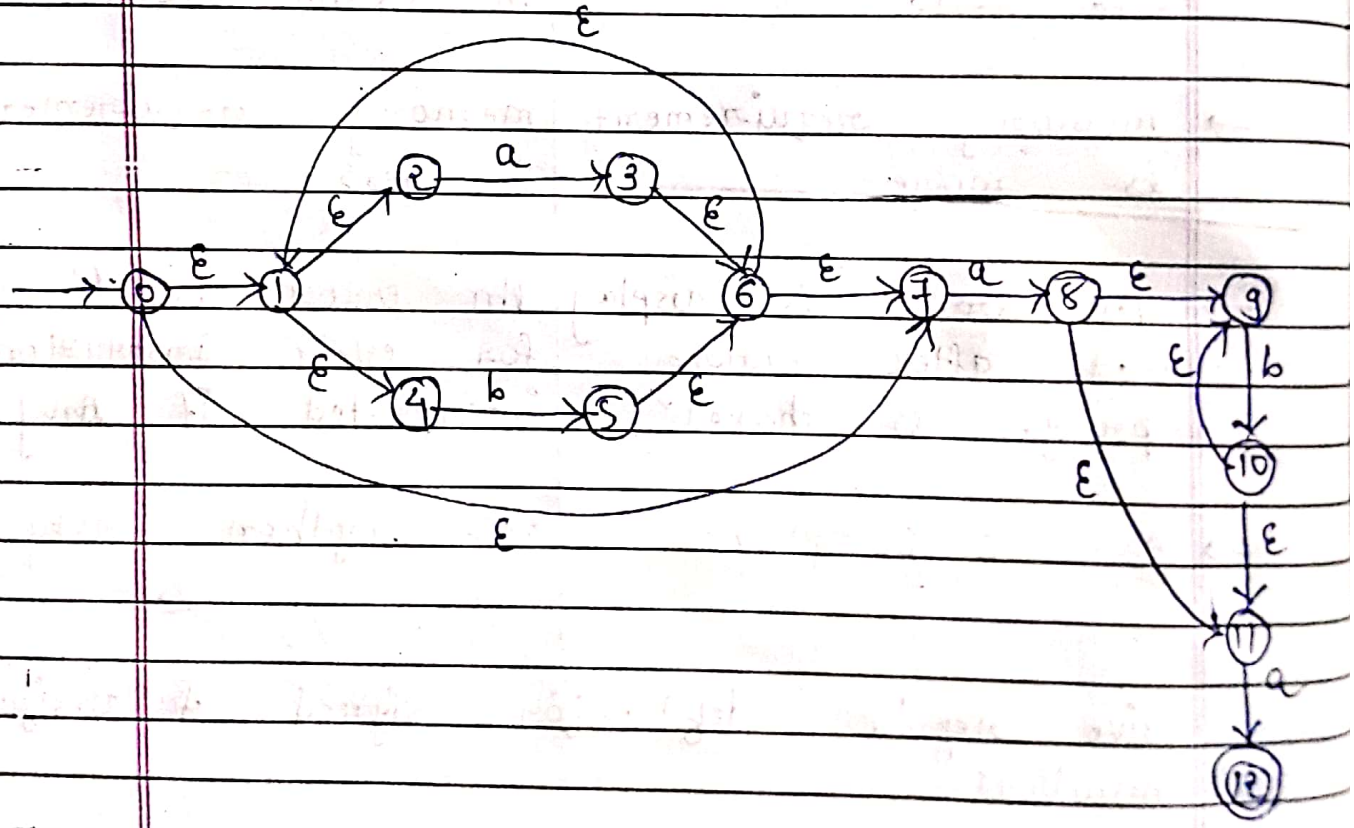
170220107030

Page No.:
Date: / /

(*) Unsigned number :

→ It doesn't have any sign there can contain only magnitude of the numbers so representation of unsigned binary numbers are positive numbers only.

Que-9) Draw NFA from regular expression using Thomson's construction & convert it into DFA.



ε-closure {0} = {0, 1, 2, 4, 7} — (A)

ε-closure (A, a) = ε-closure ({0, 1, 2, 4, 7}, a)
= ε-closure (3, 8)
= {1, 2, 3, 4, 6, 7, 8, 9, 11} — (B)

170220107030

Page No.:
Date: / /

ε-closure (A, b) = ε-closure ({0, 1, 2, 4, 7}, 6)
= ε-closure (5)
= {1, 2, 4, 5, 6, 7} —— (C)

ε-closure (B, a) = ε-closure ({1, 2, 3, 4, 6, 7, 8, 9, 11}, a)
= ε-closure (3, 8, 12)
= {1, 2, 3, 4, 6, 7, 8, 9, 11, 12} —— (D)

ε-closure (B, b) = ε-closure ({1, 2, 3, 4, 6, 7, 8, 9, 11}, b)
= ε-closure (5, 10)
= {1, 2, 4, 5, 6, 7, 9, 10, 11} —— (E)

ε-closure (C, a) = ε-closure ({1, 2, 4, 5, 6, 7}, a)
= ε-closure (3, 8) —— (H)

ε-closure (C, b) = ε-closure ({1, 2, 4, 5, 6, 7}, b)
= ε-closure (5) —— (C)

ε-closure (D, a) = ε-closure ({1, 2, 3, 4, 6, 7, 8, 9, 11, 12}, a)
= ε-closure (3, 8, 12) —— (D)

ε-closure (D, b) = ε-closure ({1, 2, 3, 4, 6, 7, 8, 9, 11, 12}, b)
= ε-closure (5, 10) —— (E)

ε-closure (E, a) = ε-closure ({1, 2, 4, 5, 6, 7, 9, 10, 11}, a)
= ε-closure (3, 8, 12) —— (D)

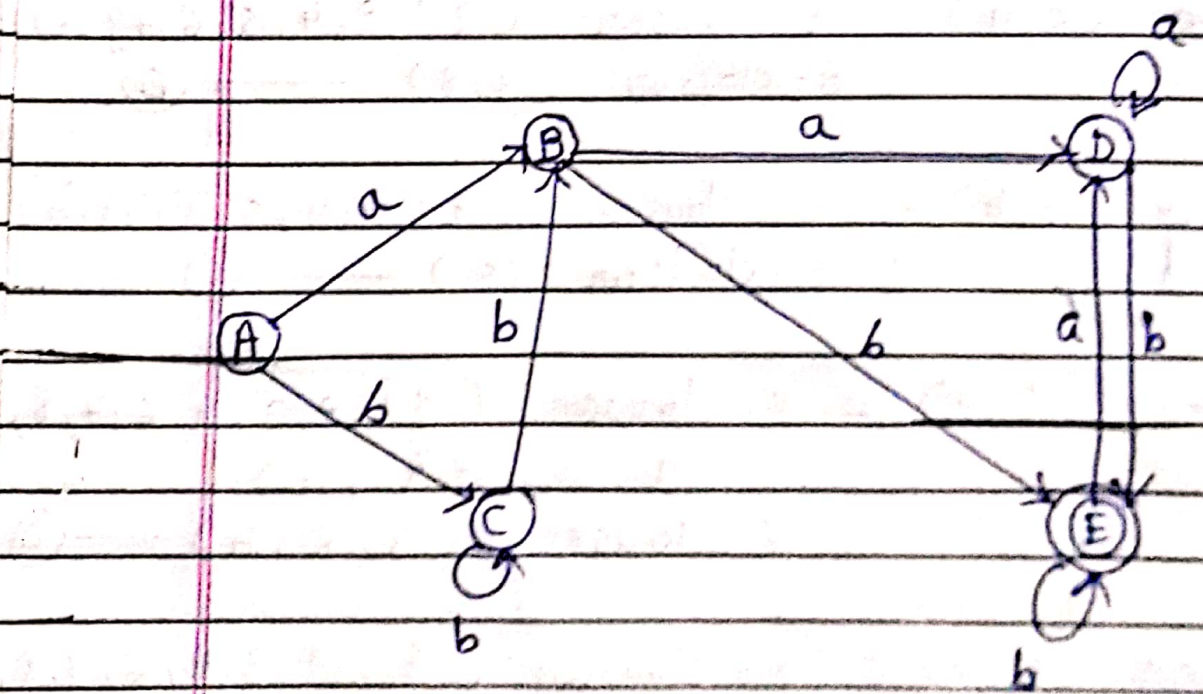
170220107030

ε-closure (E, b) = ε-closure (1, 2, 4, 5, 6, 9, 10, 11, 7, b)

= ε-closure (5, 10) ——— (F)

⊛ transition table :-

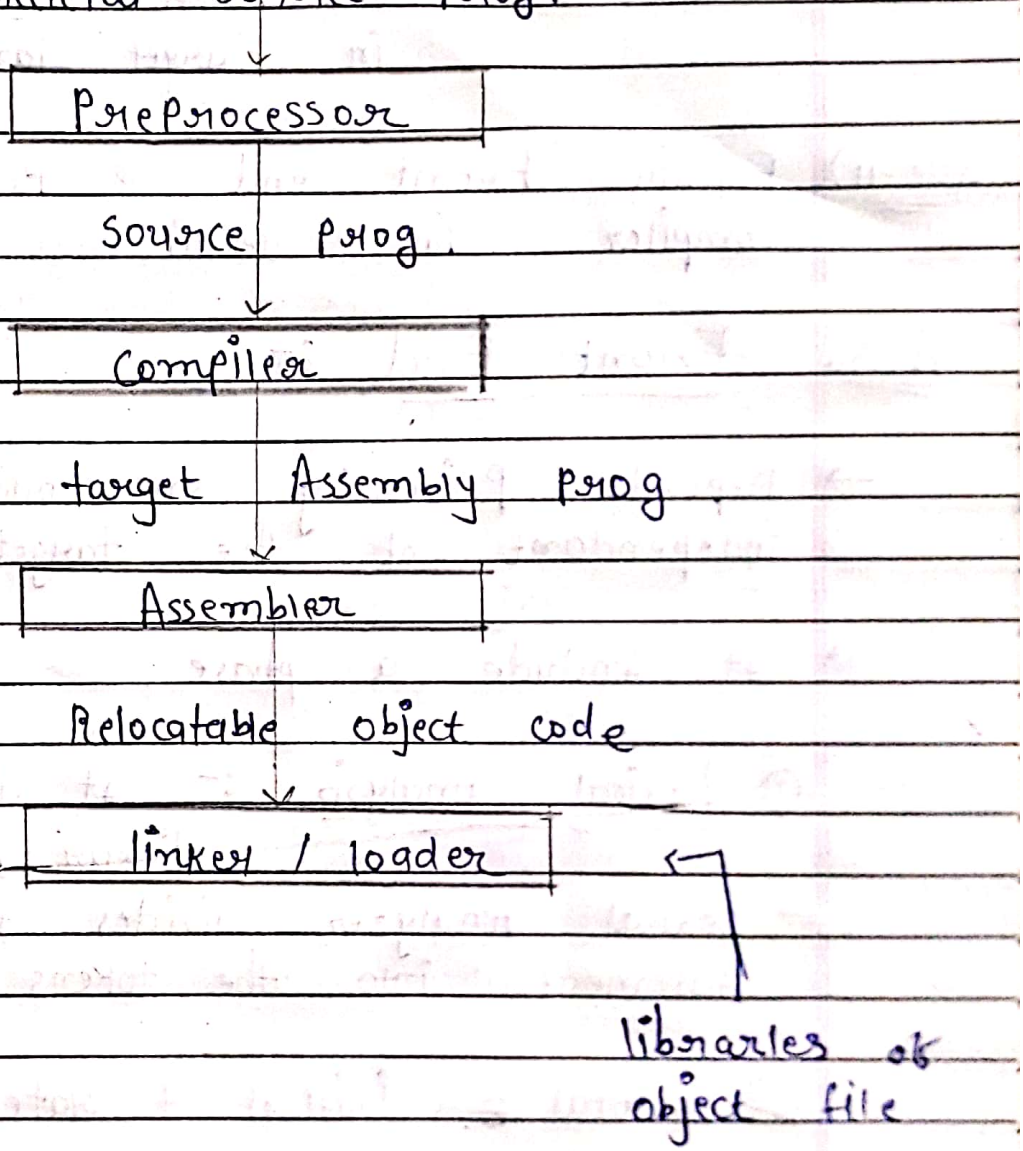
State	Input	
	a	b
A	B	C
B	D	F
C	B	C
D	D	E
E	D	F



Page No.:
Date: / /

Que-10 List the cousins of compiler & explain the role of any one of them.

Ans:-



(*) Cousins of compiler,
→ Preprocessor
→ Compiler
→ Assembler
→ Linker / loader

Page No.:
Date: / /

(6) Compiler :- It is a program that reads a program written in source lang. & translates it into an equivalent program in target language.

Que-11) Explain Front end & Back end of Compiler in detail.

Ans:- Front End :-

→ Depends primarily on source lang. & largely independant of the target Machine.

→ It Include a phase

① Lexical Analysis :- It also called as linear Analysis or Scanning

⇒ Lexical Analyzer divides the given source statement into the tokens.

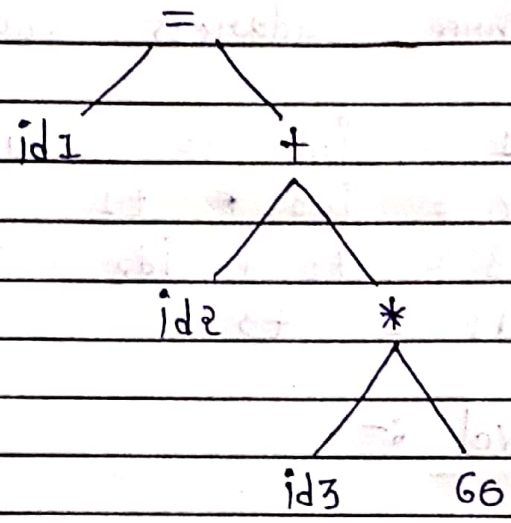
ex. total = initial + rate * 60

total (identifier)
= (Assignment symbol)
initial (identifier)
+ (operator)
rate (identifier)
* (operator)
60 (constant)

Page No.:
Date: / /

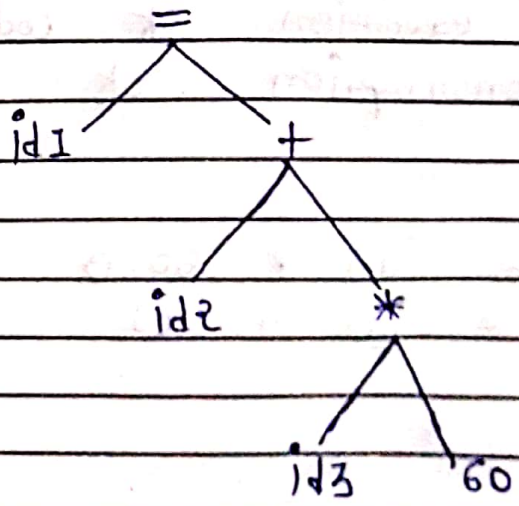
② Syntax Analysis :- It is also called Parsing.

⇒ It generates syntax tree if code is error free.



③ Semantic Analysis :- determines the meaning of a source string.

⇒ for ex, matching of paranthesis in expression, matching of if ... else statement that are type Compatible, checking the scope of operation.



. int to real

Page No.:
Date: / /

170220107030

④ Intermediate code generation :-

⇒ It should be easy to produce & translate into target program.
⇒ Intermediate form can be represented using three address code.

exp
$t1 = int\ to\ real\ (60)$
$t2 = id3 * t1$
$t3 = t2 + id2$
$id1 = t3$

(*) Back End :-

→ Depends on target Machine & don't depend on Source program.

→ It Include two phase.

① code optimization :

⇒ It Improve the Intermediate code
⇒ faster execution of code or less consumption of Memory

exp
$t1 = id3 * 60.0$
$id1 = id2 + t1$

170220107030.

Page No.:
Date: / /

② code generation :

⇒ The Intermediate code Instructions are translated into sequence of Machine Instruction.

eg

```
MOVF  id3 , R2
MULF  #60 , R2
MOVF  id2 , R1
ADDF  R2 , R1
MOVF  R1 , id1
```

~ * ~