Dated
9. Dec. 10

Chapter No. 3

SYNTAX DIRECTED TRANSLATION (SDT)

G
+
Semantic rules
(or)
Translation rules

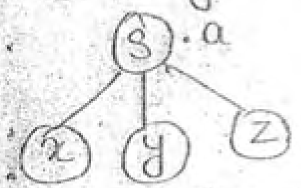
$\Longrightarrow$ SDT

Syntax tree
or
Parse tree

Attribute

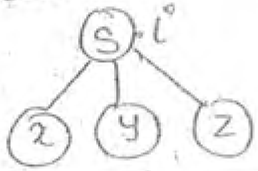
Synthesized attribute

$S \rightarrow xyz$



$\boxed{S.a = f(x.a \mid y.a \mid z.a)}$

Inherited attribute



$\boxed{y.i = f(x.i \mid S.i \mid z.i)}$

Applications of syntax directed translations

* Converting the given infix expression to postfix expression
* evaluating the given infix expression.
* Binary to decimal conversion.
* Creating syntax tree
* Creating directed acyclic graph
* to generate intermediate code
* Storing the data into symbol table

Construct Syntax Directed Translation (SDT) to convert the given the infix expression to postfix expression

I/P: 2+3 * 4

O/P: 2 3 4 * +

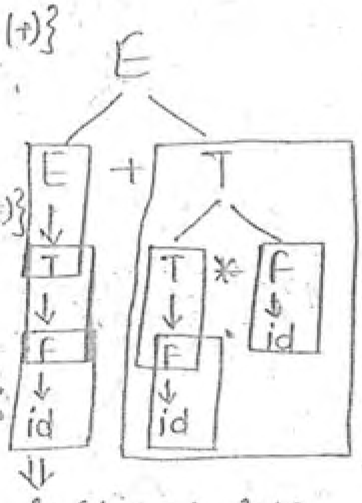
Grammar:— E → E+T {printf (+)}

| T {–}

T → T * F {printf (*)}

| F {–}

F → id {printf (id)}



F → id {printf (id)}

Q) Construct SDT to find out no. of reductions to evaluate the given infix expression:-

i/p = 2+3 * 4

O/P = 8

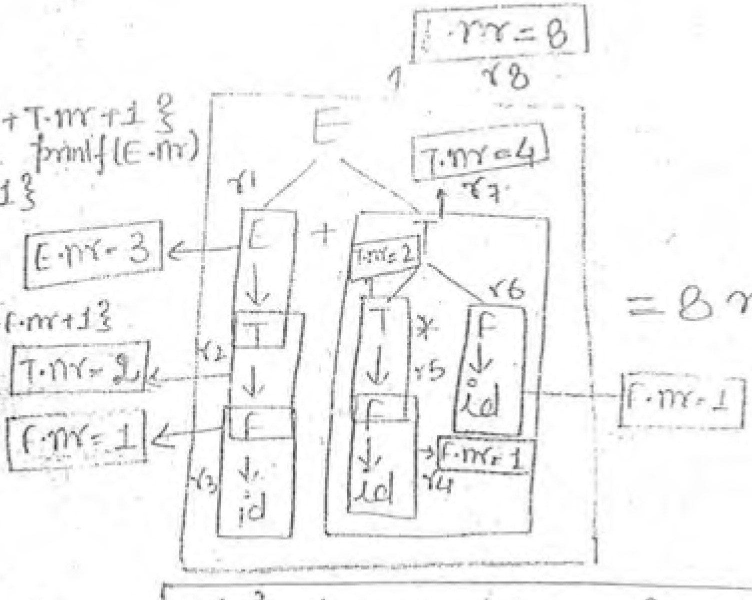
E → E+T {E.nr = E.nr + T.nr + 1 printf (E.nr)}

| T {E.nr = T.nr + 1}

T → T * F {T.nr = T.nr + F.nr + 1}

| F {T.nr = F.nr + 1}

F → id {F.nr = 1}



= 8 reductions

attribute = nr = synthesized attribute

Q) Construct SDT to evaluate the given infix expression-

I/P = 2+3 * 4

O/P = 14

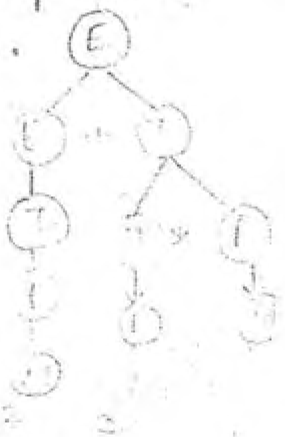
E → E₁+T {printf (E₁.val + T.val)}

| T {E.val = T.val}

T → T₁ * F {T.val = T₁.val * F.val}

| F {T.val = F.val}

F → ...



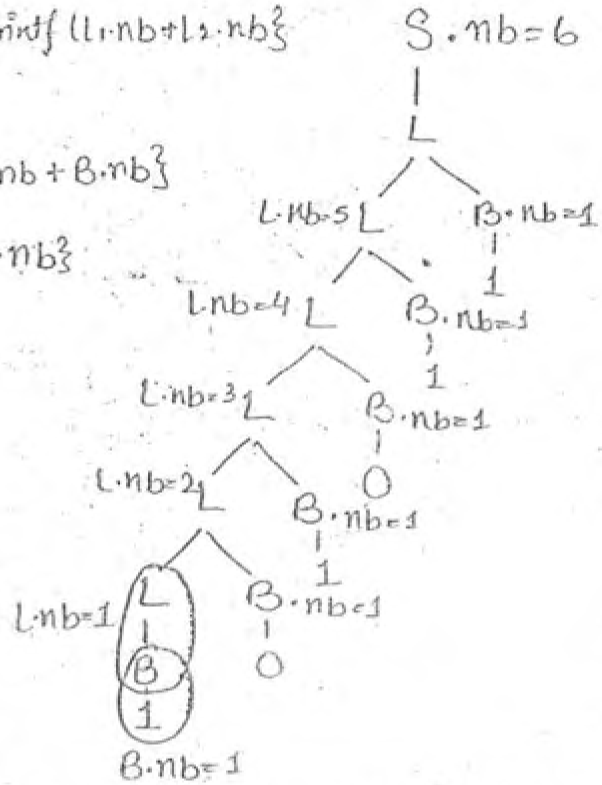
1Q. Construct SDT to find the bits in the given binary number.

Soln I/P: 101011 | 1011.111
O/P: 6 | O/P = 7

S → L | L.L {printf(L1.nb + L2.nb)}
↓
{printf(L.nb)}

L → LB {L.nb = L.nb + B.nb}
| B {L.nb = B.nb}

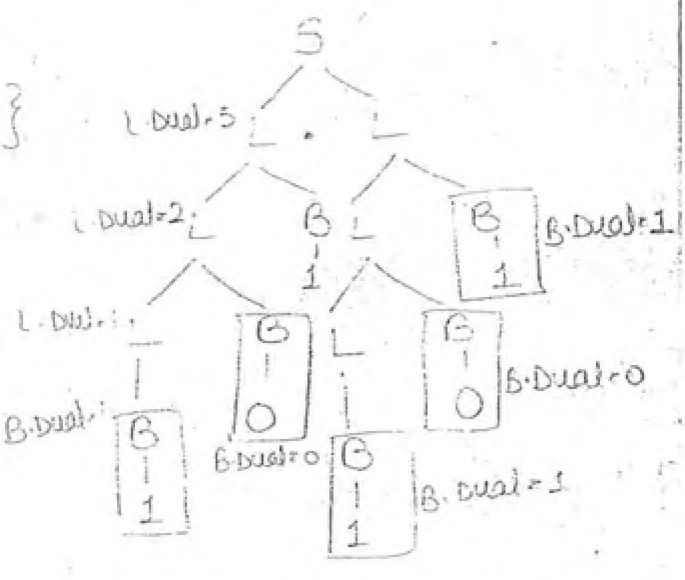
B → 0 {B.nb = 1}
| 1 {B.nb = 1}



2Q. Construct a SDT to convert given decimal no into binary number:-
I/P: 101.101
O/P: 5.625

Soln S → L / L.L ⇒ {printf(L1.DV + $\frac{L_2.DV}{2^{L_2.DV}}$)}
↓
printf{(L.DV)}
L → L B {L.nb = L.nb + B.nb ; L.DV = 2 * L.DV + B.DV}
| B {L.DV = B.DV ; L.nb = B.nb}

B → 0 {B.nb = 1 ; B.Dual = 0}
| 1 {B.Dual = 1 ; B.nb = 1}

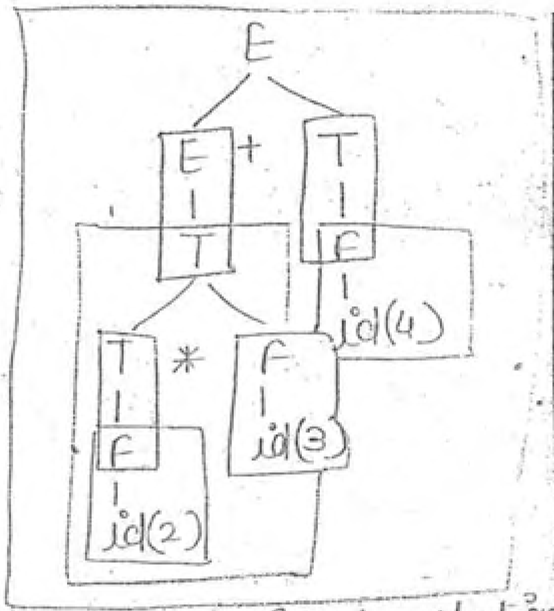


3Q. Construct SDT to convert the given infix expression into prefix expression:-
I/P:- 2 * 3 + 4
O/P:- + * 2 3 4

Sol^n $E \to E + T \Rightarrow \{printf(+)\}\ E+T$
$\quad | T$

$T \to T * F \Rightarrow \{printf(*)\}\ T*F$
$\quad | F$

$F \to id \Rightarrow \{printf(id)\}$



Q.N. [GATE] Consider the grammar with the following translation rules:- $ E as the start symbol-

$E \to E_1 \# T\ \{E.val = E_1.val * T.val\}$

$\quad | T \quad \{E.val = T.val\}$

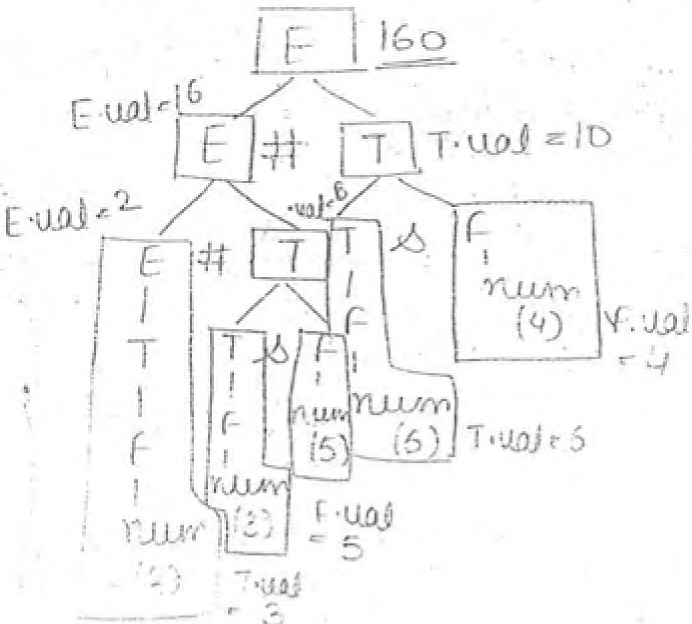
$T \to T_1 \$ F\ \{T.val = T_1.val + F.val$

$\quad | F\ \{T.val = F.val\}$

$F \to num\ \{F.val = num\}$

Compute the E.val for the root of the parse tree for the expression-
$\boxed{2 \# 3 \$ 5 \# 6 \$ 4}$

Sol^n



$\boxed{E.val = 160}$ Ans

4.2 $E \rightarrow E \# T \{E.val = E_1.val * T.val\}$ |

$\quad | T \ \{E.val = T.val\}$

$T \rightarrow T \$ F \{\text{——}\}$ $T.val = T.val - F.val$

$\quad | F \ \{T.val = F.val\}$

$F \rightarrow num \ \{F.val = num\}$

Q4.0) If the expression 8#12$4#16$12#4$2 is evaluated to 512, which one of following is correct -

(a) $T.val = T_1.val * F.val$

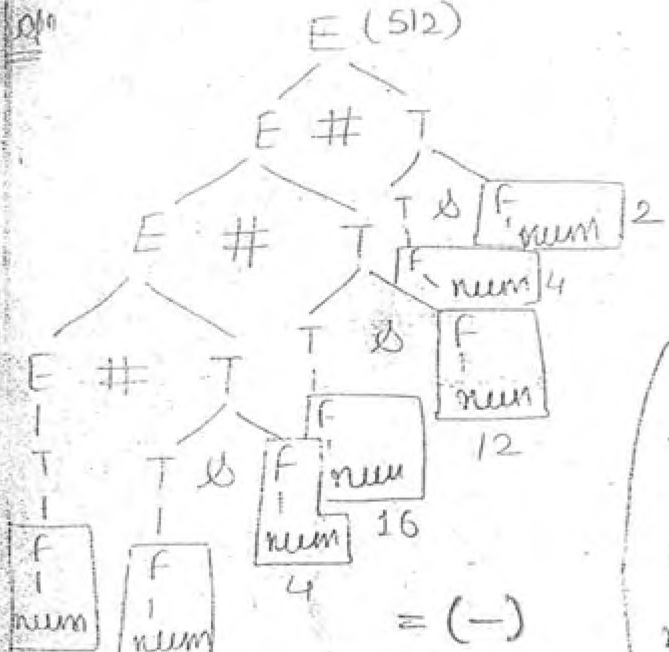
(b) $T.val = T_1.val + F.val$

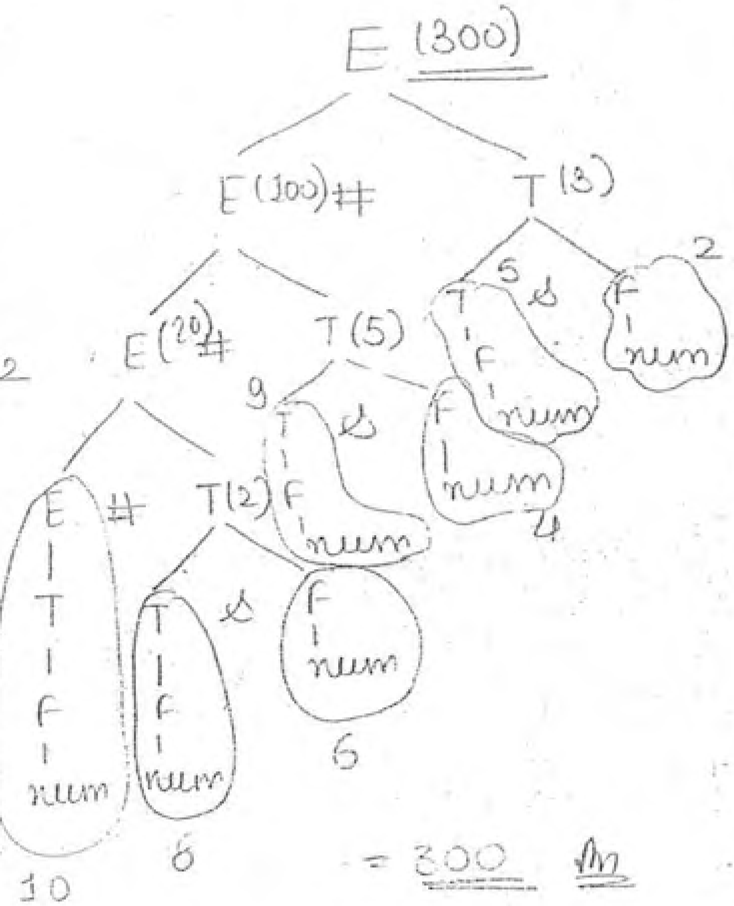
(c) $T.val = T_1.val / F.val$

(d) None

Ans

QN(b) Compute 10#8$6#9$4#5$2





= 300 Ans

QN If the given grammar is -

$S \rightarrow TR$

$R \rightarrow + T \{printf('+')\} R \mid \epsilon$

$T \rightarrow num \{printf(num)\}$

If the I/P is- 9+5+2, what will be the O/P-

a) 9+5+2

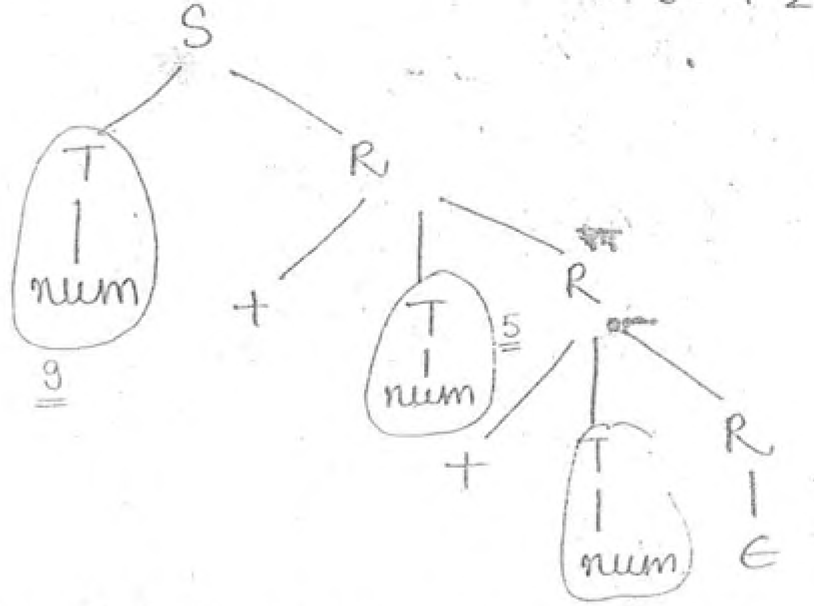
b) 95+2+ ✓

c) 952++

d) ++952

⇒ 95+2+

Sol^n



Q) Construct the SDT to store type info into symbol table.

I/P: int x, y, z;

O/P:

V.name	V.type
x	int
y	int
z	int

Sol^n $D \rightarrow D_1, id$ $\boxed{\{D.type = D_1.type;\ addtype(id, D_1.type)\}}$

$| \ T \ id$ $\boxed{\{D.type = T.type;\ addtype(id, T.type)\}}$

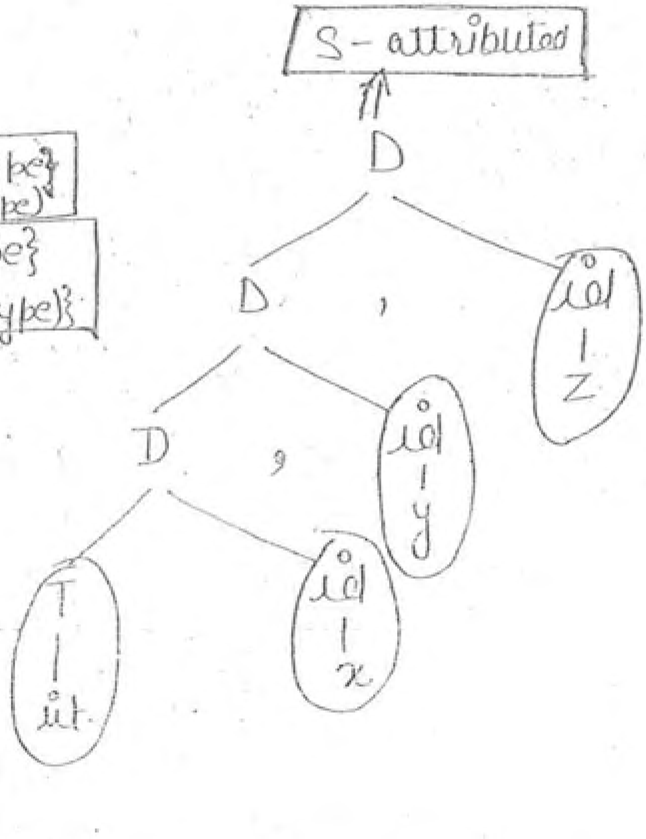
$T \rightarrow int$ $\boxed{\{t.type = int\}}$

$| \ float$ $\boxed{\{t.type = float\}}$

$| \ char$ $\boxed{\{t.type = char\}}$

$id \rightarrow a|b|c$

$|z$



9

SDT

- S-attributed SDT — Synthesized (Postorder) attribute
- L-attributed SDT — Left to right

Problems Using Inherited attributes (Preorder evaluation)

QH $S \rightarrow T\,I$ $\boxed{\{I.i = T.type\}}$

$T \rightarrow int$ $\boxed{\{T.type = int\}}$

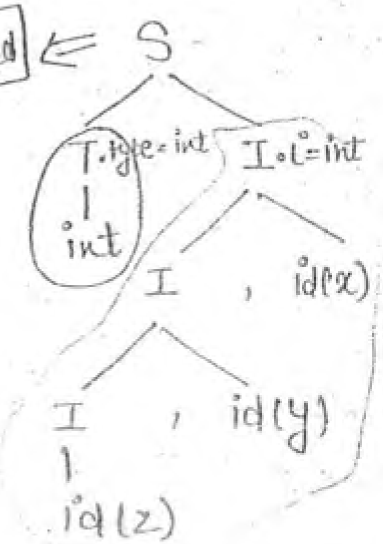
| char

| float

$I \rightarrow I_1, id$ $\boxed{\{A\ I_1.i = I.i\}\ \{add\ type(id, I_1.type)\}}$

id $\boxed{\{addtype(id, I_1.type)\}}$

L-attributed ⇐



∴ If one required value is on L.H.S and other value is on R.H.S., no method other than inherited attribute.

SDT

- S-attributed SDT
- L-attributed SDT

S-attributed SDT	L-attributed SDT
we will use only synthesizer attribute	① we can use both synthesizer and inherited attribute. But in the case of inherited attribute, only values from root or the left siblings can be taken not from right sibling
Semantic rules will be placed after end of the production	② We can place the semantic rule anywhere at the right hand side of production

evaluation is ③ The order of evaluation - Preorder.

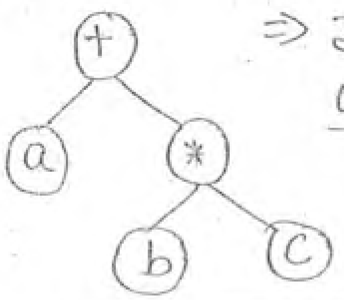
attributed definitions are also S-attributed definition

$\boxed{S\text{-attribute} \subseteq L\text{-attribute}}$

QH. Construct SDT to construct syntax tree for the given infix expression:-

I/P: a+b*c

O/P:



⇒ In the syntax tree leaf nodes are Operands and intermediate nodes are operator.

⇒ makenode(lp, id, rp) = Creating a node

⇓

will return the location, where the node is created.

Grammar

E → E_1 + T {mknode(E_1.ptr, +, T.ptr)}

| T {E.ptr = T.ptr}

T → T_1 * F {T.ptr = mknode(T_1.ptr, *, F.ptr)}

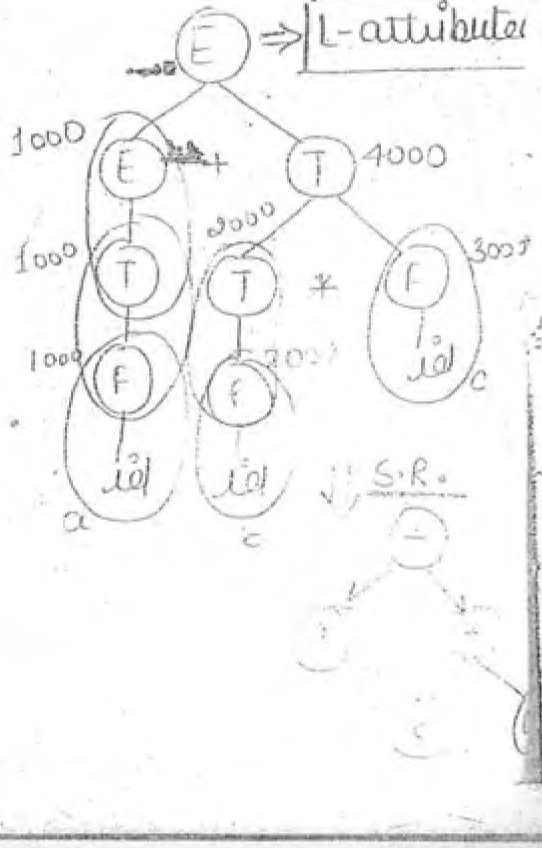
| F {T.ptr = F.ptr}

F → id {F.ptr = mknode(N, id, N)}

S-attribute

⇑

⇒ L-attributed



1. Give a grammar to reverse the given infix expression-

I/P: (a+b) * (c+d)

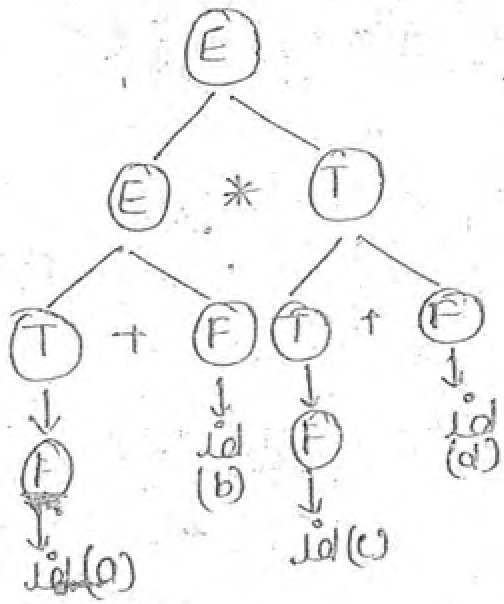
O/P: (d+c) * (b+a)

Soln

$E \rightarrow E * E \mid (T)$

$T \rightarrow T + F \mid F$

$F \rightarrow id$



Semantic rules

$E \rightarrow E * E$

$\rightarrow (T)$

$\rightarrow T + F$

$\mid F$

$F \rightarrow id$ {F.val = id}

II. Construct SDT to generate three address code (Intermediate code) for the given infix expression:-

i/p: $x = a + b * c$ → newtemp() = Create temporary variable

o/p: $t_1 = b * c$

$t_2 = a + t_1$ → gen(t = b*c)

$x = t_2$

Soln

$S \rightarrow id = E$ {gen(id = E.val)}

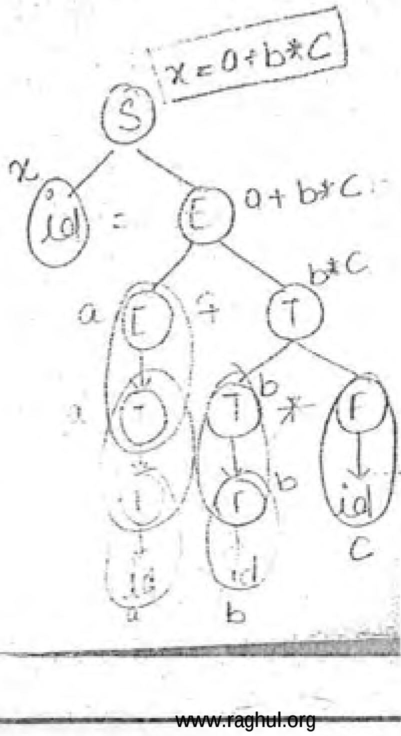
$E \rightarrow E + T$ {Q = newtemp(); gen(Q = E1.val + T.val); E.val = Q}

$\mid T$ {E.val = T.val}

$T \rightarrow T_1 * F$ {P = newtemp(); gen(P = T1.val * F.val); T.val = P}

$\mid F$ {T.val = F.val}

$F \rightarrow id$ {F.val = id}



QN GATE

Consider the SDT shown below:-

$S \rightarrow id = E$ { gen (id.place = E.place) }

$E \rightarrow E_1 + E_2$ { t = newtemp();
gen(t = E_1.place + E_2.place)
E.place = t; }

$E \rightarrow id$ { E.place = id }

Here gen is a function that generates the O/P code and new temp(); is a function that returns the name of new temporary variable at every call. Assume that t_i are the new temporary variable name, generated by new temp. For the statement $\boxed{x = y + z,}$ of the three address code generated by the above SDT is-

a) $x = y + z$

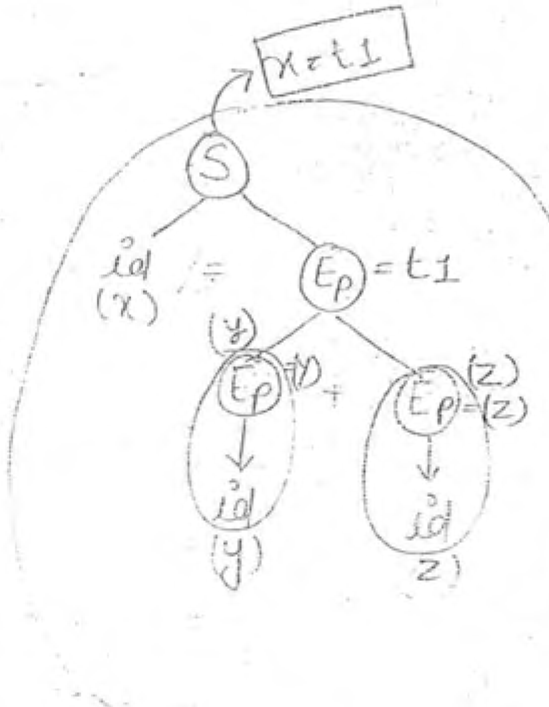
b) $t_1 = y,\ t_2 = t_1 + z$
$x = t_2$

c) $t_1 = y + z$
$x = t$

d) $t_1 = y,\ t_2 = z$
$t_3 = t_1 + t_2,\ x = t_3$.

Soln $t_1 = y + z$

$\boxed{x = t_1}$



1. Consider the following SDT:-

E → number {E.val = number}

| E+E {E.val = E1.val + E2.val}

| E*E {E.val = E1.val * E2.val}

YACC

⇓

$\boxed{\text{Yet Another Compiler Compiler}}$

Soln I/P: 3*2+1

YACC (Give more priority to shift (Push), rather than reduce (pop))

*	+	

(3, (2, 1, +) *)

nd QH a) The above grammar and semantic rule is given to YACC tool for parsing and evaluating arithmetic expressions, which one of the following is true, about the action of YACC for given grammar-

i) It detects recursion and eliminate

ii) It detects reduce-reduce conflict and resolves

iii) It detect shift-reduce conflict and resolve the conflict and resolves in favour of shift over reduce.

iv) resolves favour of reduce over shift

b) Assume the conflict in QH(a), what will be the precedance and associativity for the expression- $\boxed{3*2+1}$

i) equal precedance and left associative, evaluated to 7.

ii) Equal precedance and right associativity, evaluated to 9.

YACC tool = LALR(1) parser generator

Parser → no multiple value entry

⇓

LL(1) or LR(1) ⇒ LL(1). Because in LR(1) there is YACC tool.

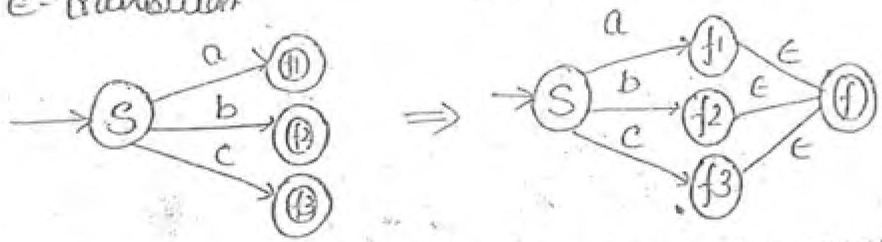
Dated
28.Dec.10

FA → RE

Steps

Step-1 If more than 1 final state is there, make it as single final by adding ε-transition

Exp:-



* A NFA with more than one final state can be converted into equivalent NFA with single final state, but it is not possible incase of DFA.

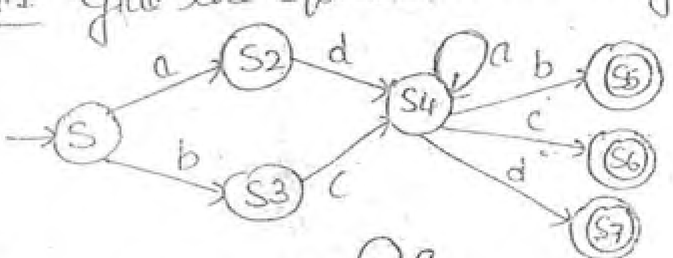
Step-2 If more than 1 edge going in same direction make it as single edge and label with union with symbol.

Step-3 If more than one edge is going in same direction one after another, making it as single edge, with the label of concatenation of symbols.

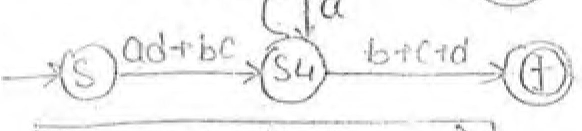
Exp:-

Step-4 State Elimination method

QN.1 Give the equivalent the regular expression:-

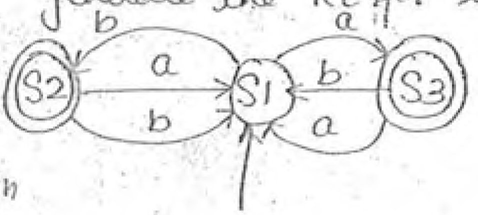


Soln

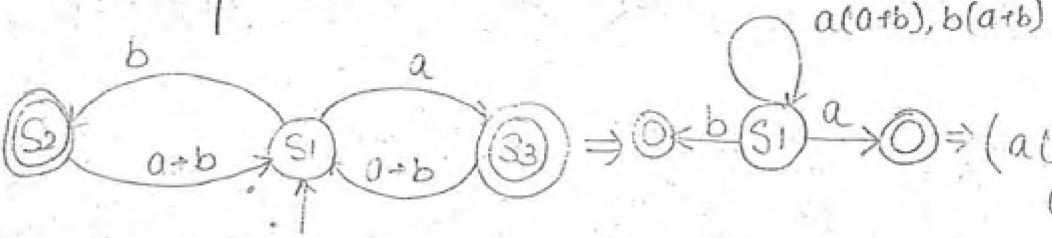


= $\boxed{(ad+bc)\, a^* \,(b+c+d)}$ Ans

Q.N Generate the RE for the following automata :-



Sol^n



$\Rightarrow (a(a+b)+b(a+b))^*(a+b)$.

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

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

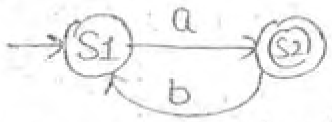
⇓

Even length string followed by a or b

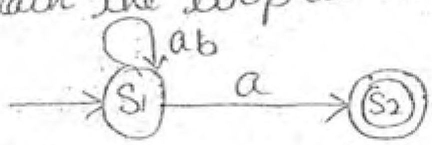
⇓

Odd length string **Ans**

Q.N Give the RE for the following finite automata :-

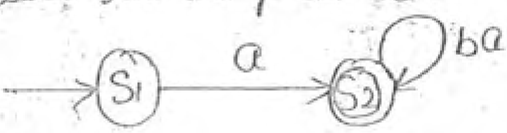


Maintain the loop at S1 :-



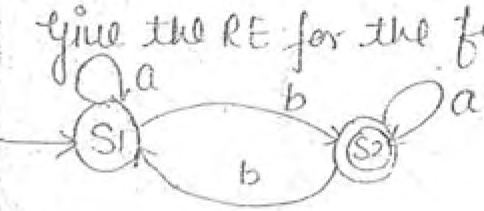
$\Rightarrow (ab)^*a$

Maintain the loop at S2 :-

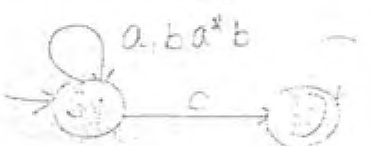


$\Rightarrow a(ba)^*$

Q.N Give the RE for the following finite automata :-



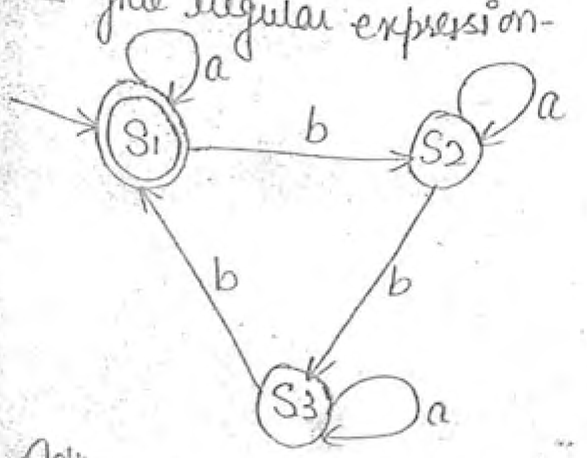
Sol^n



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

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

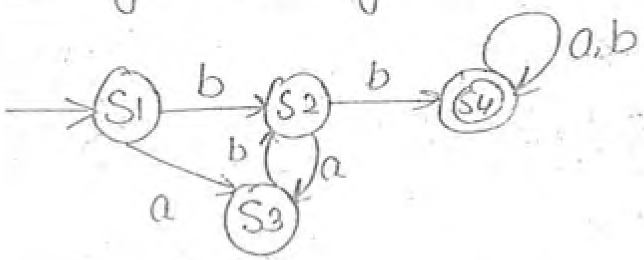
QN Give regular expression-

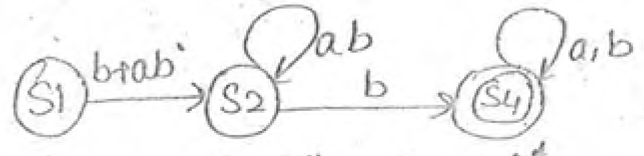


Soln

$$\left(a+(ba^*ba^*b)\right)^*$$

QN Give the regular expression:

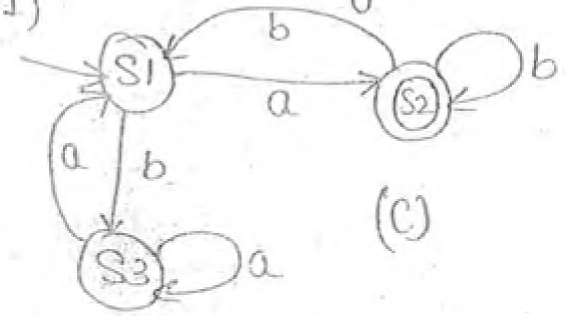




$$= (b+ab)(ab)^* \, b \, (a+b)^*$$

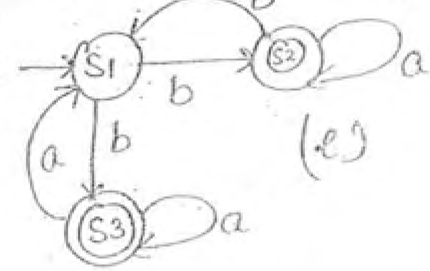
QN Match each of the NFA, with corresponding matching option-

1)



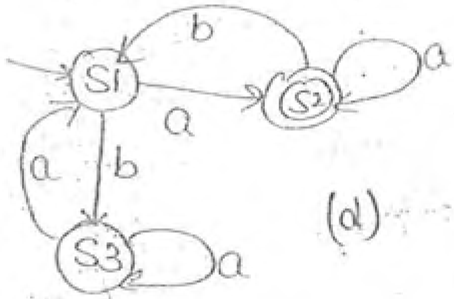
(C)

(5)



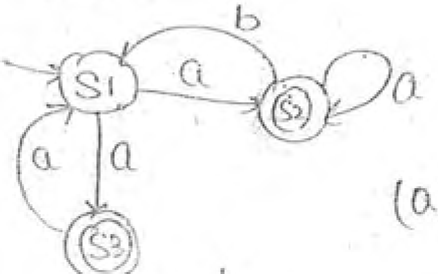
(e)

2)



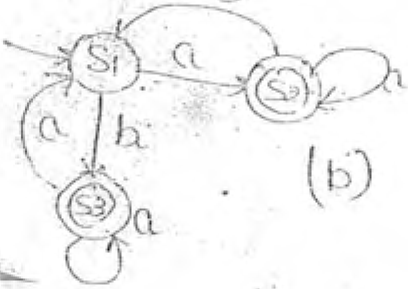
(d)

3)



(a)

4)



(b)

a) $(aa^*b+ba^*b)^* \, ba^*$

b) $(aa^*a+aa^*b)^* \, aa^*$

c) $(ba^*a+ab^*b)^* \, ab^*$

d) $(ba^*a+aa^*b)^* \, aa^*$

e) $(ba^*a+ba^*b)^* \, ba^*$

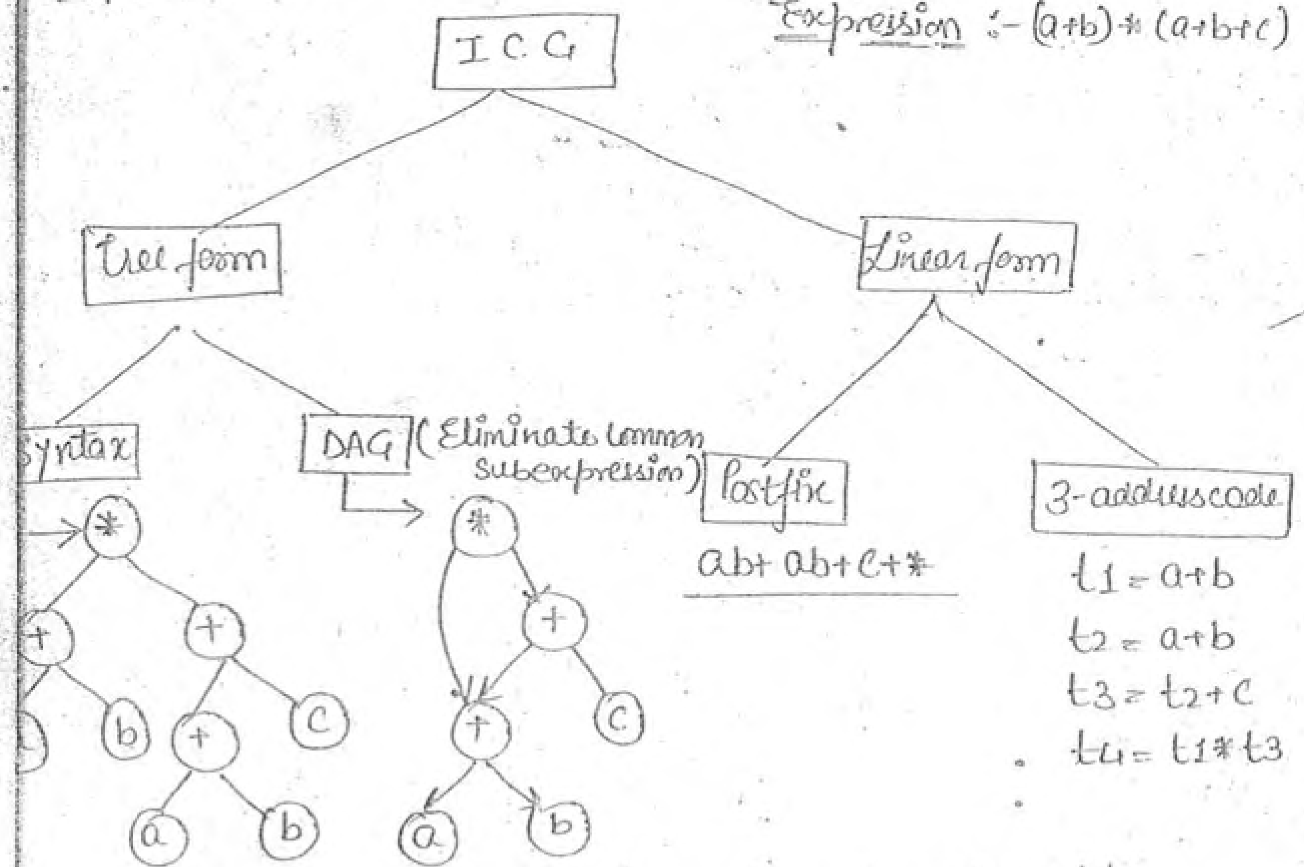
Dated
3 Dec.10

| Chapter No. 4 |

Intermediate Code Generation

Representation of intermediate code generation

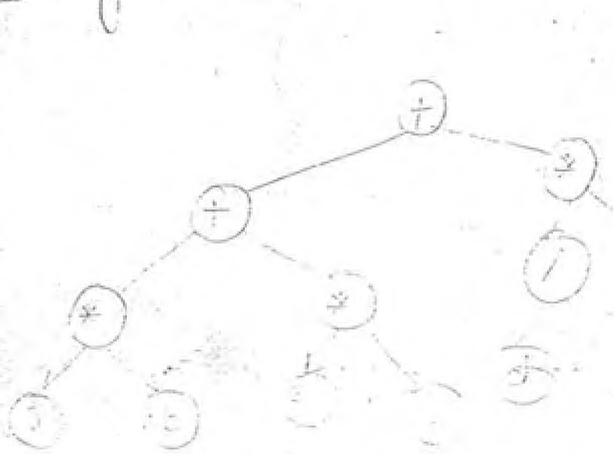
Expression :- $(a+b)*(a+b+c)$



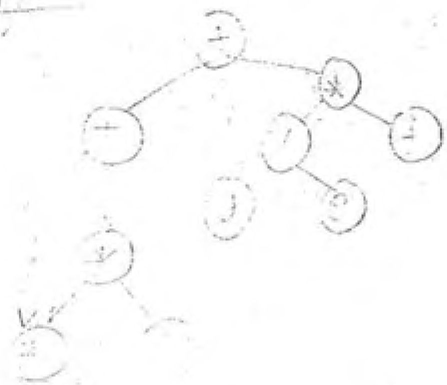
DAG:- atleast one node with indegree '0' and outdegree '0'.

Q.No.9 $(a*b)+(a*b*c)+d/e*f$

Sol^n Syntax tree



DAG



Postfix

ab* ab*c*+ de/f*+

3-address code

$t_1 = a*b$

$t_2 = a*b$

$t_3 = t_2 * c$

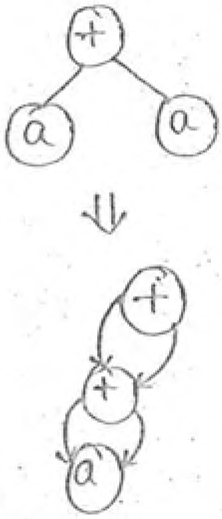
$t_4 = t_1 + t_3$

$t_5 = d/e$

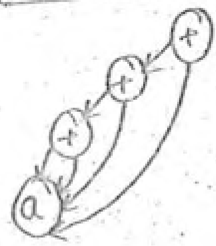
$t_6 = t_5 * f$

$t_7 = t_4 + t_6$

DAG - (a+a)+(a+a)



a+a+a+a



(a+a)+(a+a+a)

Types of three address code

1) $x = y \ op \ z$
2) $x = op \ y$
3) $x = y$
4) $x = *y$
5) $x = \& y$
6) $x = a[i] \Rightarrow *(a+i)$
7) $a[i] = x$
8) goto L (unconditioned jump)
9) if $x<y$ goto L

No three address code

- $x = a[i,j]$
- $x = fun(a,b)$

Q) Construct three address code for the following expression

if a<b then t=1 else e=0

Soln It is not a three address code.

⇓ Conversion in three address code

i) if a<b goto <u>i+3</u>

(i+1) e=0

(i+2) goto <u>i+4</u>

(i+3) t=1

(i+4) —

Back patching (filling gaps)

Q) If a<b && c>d then t=1 else e=0

Soln It is not in three address code.

⇓ Conversion in three address code

i) if a<b goto <u>i+1</u>

(i+1) if c>d goto <u>i+4</u>

(i+2) e=0

(i+3) goto <u>i+5</u>

(i+4) t=1

(i+5) —

i) if a<b goto <u>i+2</u>

(i+1) goto <u>i+3</u>

(i+2) if c>d goto <u>i+5</u>

(i+3) e=0

(i+4) goto <u>i+6</u>

(i+5) t=1

(i+6) —

Q) Construct three address code for 'while' statement in C language.

Soln i=0

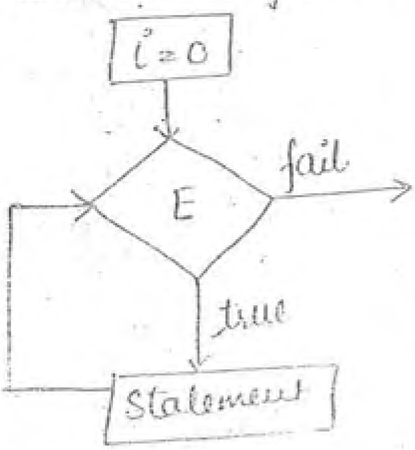
while(i<10)

{

x=a+b*c;

i++;

}



<u>Three address code (condition)</u>

i=0

S) if i<10 goto <u>S+2</u> (if true)

S+1) goto <u>S+7</u> else outside

S+2) t1 = b*c

S+3) t2 = a+t1

S+4) x = t2

S+5) i = i+1

S+6) goto <u>S</u>

S+7) —

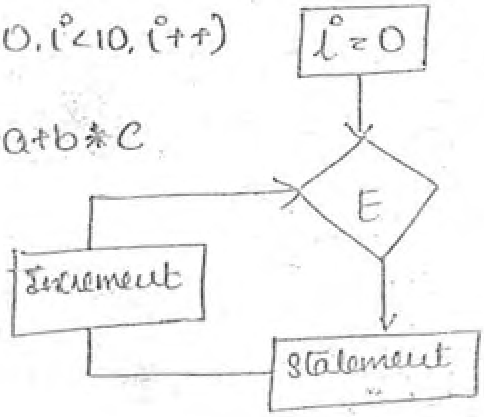
QN Construct three address code for 'for' loop in c language.

Soln for($i=0, i<10, i++$)

{

$x = a+b*c$

}



S+0) $i=0$

S+1) if $i<10$ goto S+2

S+1) goto S+7

S+2) $t1 = b*c$

S+3) $t2 = a+t1$

S+4) $x = t2$

S+5) $i = i+1$

S+6) goto S

S+7) —

for ($i=0; i<10; i--$)

$a = b+c$;

}

If $i=0$ goto S+1

$t_1 = a+1$

$a = t_1$

QN Construct a three address code for switch statement in 'C' language

Soln $i=1$

switch (i)

{

Case 1: $x_1 = a_1+b_1*c_1$

break;

Case 2: $x_2 = a_2+b_2*c_2$

break;

default: $x_3 = a_3+b_3*c_3$

}.

Three address code

$i = 1$

S) if($i==1$) goto Case 1

S+1) if($i==2$) goto Case 2

S+2) $t1 = b3*c3$

S+3) $t2 = a3+t1$

S+4) $x3 = t2$

S+5) ——

Case 1:) $t1 = b_1*c_1$

$t2 = a_1+t1$

$x_1 = t2$

goto S+5

Case 2:) $t1 = b_2*c_2$

$t2 = a_2+t1$

$x_2 = t2$

goto S+5

Q. Construct three address code for. $x = a[i][j]$, suppose a [10][20]

Soln It is not a three address code.

⇓ Conversion in three address code.

$x = a[i][j] = *(*(a+i)+j)$

$t_1 = i * 20$ a[5,10]

$t_2 = t_1 + j$

$x = a[t_2]$

$5*20 = 100$
$+10$
110

Representation of three address code

1) Quadraples

2) Triples

3) Indirect Triples

Expression :- $-(a+b)*(a+b*c)$

1) Quadraples → Advantage - can move the result. Disadvantage - More space

S.No.	OP	OP1	OP2	Result
1	+	a	b	t1
2	–	t1		t2
3	*	b	c	t3
4	+	a	t3	t4
5	*	t2	t4	t5

Memory (Quadraples)

OP1	OP2	result
a	b	t1
t1		t2
b	c	t3
a	t3	t4
t2	t4	t5

2) Triples

S.No.	OP	OP1	OP2
1	+	a	b
2	–	(1)	
3	*	b	c
4	+	a	(3)
5	*	(2)	(4)

Advantage :-

* Less space.

* Can't move the result at desired place. → disadvantage

3) Indirect touples

→ If there is a requirement, then we can move the result to some another location by copying the same values.

Advantages

* less space is required.

* Results can be move.

Q4 $(a+b) * (a+b+c) * d/e + f$

Soln Quadraples

S.No.	OP.	OP1	OP2	Result
1	+	a	b	t1
2	+	a	b	t2
3	+	t2	c	t3
4	*	t1	t3	t4
5	*	t4	d	t5
6	/	t5	e	t6
7	+	t6	f	t7

Triples

S.No.	OP	OP1	OP2
1	+	a	b
2	+	a	b
3	+	(2)	c
4	*	(1)	(3)
5	*	(4)	d
6	/	(5)	e
7	+	(6)	f

Indirect Triples

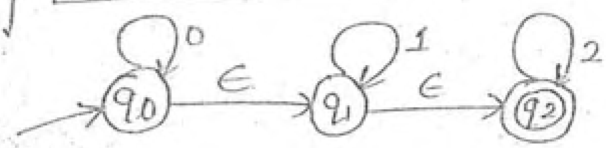
S.No.	OP	OP1	OP2	Copy
1	+	a	b	
2	+	a	b	
3	+	t2	c	
4	*	t1	t3	500
5	*	t4	d	
6	/	t5	e	
7	+	t6	f	

ted
·Dec·2010

Chapter No·5

CODE OPTIMIZATION

ϕ $\boxed{\epsilon\text{-NFA} \Rightarrow \text{NFA}}$



Sol^n Conversion

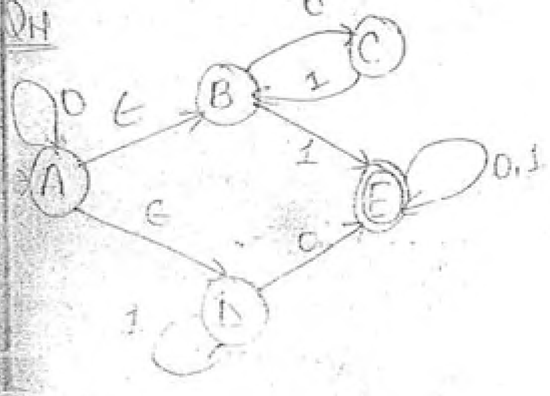
	0	1	2
$\rightarrow q_0^*$	q_0, q_1, q_2	q_1, q_2	q_2
q_1^*	ϕ	q_1, q_2	q_2
q_2^*	ϕ	ϕ	q_2

Q^N Construct NFA for following ϵ-NFA:-

Sol^n

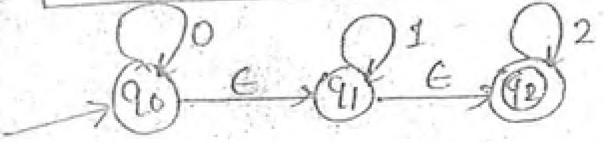
	0	1
$\rightarrow A^*$	A, B, C, D	D
B^*	C, D	D
C	ϕ	B, D
$^*D$	D	D

Q^N

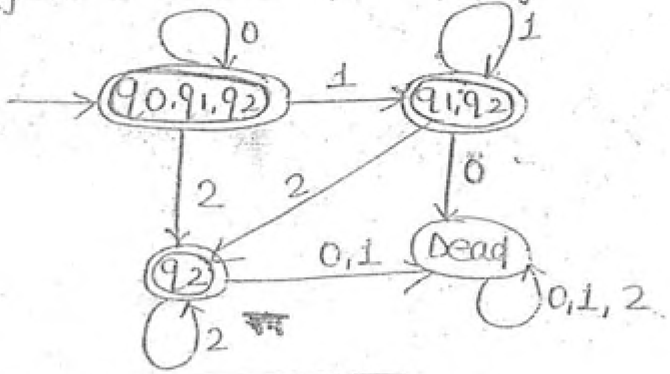


	0	1
$\rightarrow$ *A	A, B, C, D, E	D, E
B	C	E
C	ϕ	B
D	E	D
*E	E	E

ε-NFA → DFA



- Find ε-closure to starting state, then-



Quotient Operation

If L_1 is regular and L_2 is also regular, then L_1/L_2 is also regular.

$$\boxed{L_1/L_2 = \{x \mid \text{if } xy \in L_1 \text{ and } y \in L_2\}}$$

Exp:- $L_1 = \{b^2, b^4, b^6, b^8, \ldots\}$

$L_2 = \{b\}$

$L_1/L_2 = \{b, b^3, b^5, b^7, \ldots\}$

$L_3 = \{a\}$.

$L_1/L_3 = \{\ \}$

$L_2/L_1 = \{\ \}$.

Exp:- $L_1 = \{101, 011, 0010, 00\}$.

$L_2 = \{0, 1\}$

$L_3 = \{00\}$.

$L_1/L_2 = \{001, 0, 10, 01\}$.

$L_1/L_3 = \{\epsilon\}$

$L_3/L_2 = \{0\}$.

Note 1: In L_1/L_2, if L_2 contain ε, then $L_1/L_2 = L_1 \cup \{\ \}$.