Compiler Design

S.No: 25
Peska Sathwik

Analytical Day-1:-

(1) Find a no. of tokens in the following C statement is

```
int main()
{
  /* find max of a and b */
  int a=10, b=30;
  if (a<b)
    return (b);
  else
    return(a);
}
```

```
int   main ()
{
  /* find max of a and b */
  int a=10;   b=30;
  if (a<b)
    return (b);
  else
    return(a);
}
```

∴ No. of token = 31

(2) Write a regular expression to denote set of all strings over $\{0,1\}^*$ containing substring 101.

S.No: 25
P. Sathwik

Expression: $^(?=.*101).*$

Explanation:

(i) '^' starting of string

(ii) (?=.*101) for checking if "101" anywhere in the string or not

(iii) '*' matches zero or more.

(iv) $ asserts the end of string.

string. contains the substring "101"

(3) Write a regular expression that have at least two consecutive 0's (or) 1's

Expression: $^(.*00.*|.*11.*)$

Explanation:

(1) '^' starting of string

(2) (.*00.*|.*11.*) group that matches consecutive 0 or 1

(3) '|' logical or operator

This expression matches any string that atleast two characters.

④ How would you trace the program segment "a = (b+c)* (b+c) * 2" for all phases of compiler.

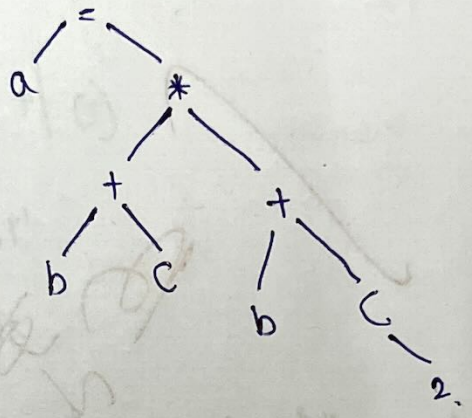
S.No: 25

P. Sathwik

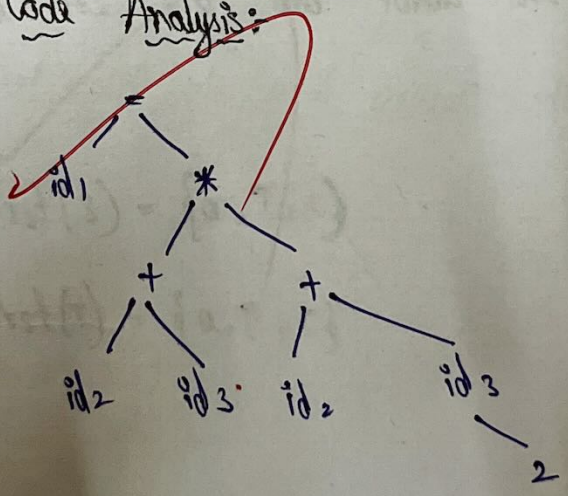
Lexical Analysis. Token:

"a", "=", "(", "b", "+", "c", ")", "*", "(", "b", "+", "c", ")", "*", "2"

Syntax Analysis:- Parse tree



Semantic Code Analysis:



Intermediate Code generator:-

$t_1 = b+c$

$t_2 = t_1 * t_1$

$a = t_2 << 1$

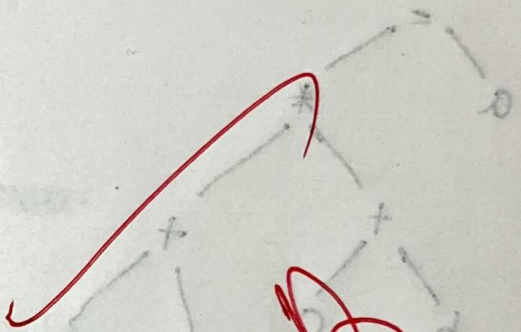
Code Optimization:

The optimized generated code is translated into machine code.

S.No: 25
P. Sathwik

Code Generator:

```
MOV   eax, b
ADD   eax, b
IMUL  eax, eax
SHL   eax, 1
MOV   a, eax
```



24/7/22

Analytical Day-2

S.No: 25

P. Sathwik

(1) Predictive Parser

$S \rightarrow a \mid \uparrow \mid (T)$ $T \rightarrow T, S \mid S$

(i) $S \rightarrow a \mid \uparrow \mid (T)$

$T \rightarrow T, S \mid S$

Step 01: Eliminate left-recursion from grammar

$S \rightarrow a \mid \uparrow \mid (T)$

$T \rightarrow ST'$

$T' \rightarrow ,ST' \mid \epsilon$

Step 02: Create the first and follow sets for each non-terminal

step ①

first(S) = {a, ↑, (}

first(T) = {a, ↑, (}

step ②

follow(S) = {$,), ,}

follow(T) = {), $, ,}

step ③ Predictive Parsing table

(ii)

	a	↑	(	,	$
S	a	↑	(		

Passing table for T:

S.No: 25
P. Sathwik

	a	↑	c	,	;	$
T	a	↑	c			
T'					•	

(2) SLR

S → cc c → cc/d

Step 01: Augment the grammar

S' → S

S → cc

c → cc/d

Step 02: Computer the closure & go to sets for item.

LR(0) item:

1. S' → .S
2. S → .cc
3. S → .cc
4. S → .d
5. c → .cc
6. c → .d

I0:

closure (I0)

1. S' → .S

Goto (I0):

Go To (I0, S) = I1

S.No: 25
P. Sathwik

1. $S' \rightarrow S$

Go To (I1):

Go To (I1, C) = I2

I2 : Closure (I2)

1. $S \rightarrow .CC$
2. $S \rightarrow .CC$
3. $S \rightarrow .d$
4. $C \rightarrow .CC$
5. $C \rightarrow .d$

Go To (I2):

Go To (I2, C) = I3

Go To (I2, C) = I4

Go To (I2, d) = I5

I3:

Closure (I3):

1. $S \rightarrow CC$
2. $C \rightarrow .CC$
3. $C \rightarrow .d$

GoTo (I3):

Go To (I3, C) = I6

I4:

Closure(I4):

1. $S \rightarrow C.C$
2. $C \rightarrow .CC$
3. $C \rightarrow .d$

Go To (I4):

Go To (I4, C) = I7

15: closure(15)

closure(b):

1. $C \to cC$
2. $C \to \cdot cC$
3. $C \to \cdot d$

Go To (16, C) = 8

. Closure(17):

$C \to cC$

closure(18)

1. $C \to d$

	c	d	$	S	C
10				shift 1	
11			Accept		
12	shift 3	shift 5		shift 4	shift 6
13	shift 3	shift 5			4
14	"	"		"	"
15			Reduce 2		
16					
17					
18	"	"			

S.No: 25
P. Sathwik

3A) Grammar

$S \rightarrow AaAb / BbBa$

$A \rightarrow \varepsilon$

$B \rightarrow \varepsilon$ is LL(1) or not?

$S \rightarrow AaAb / BbBa$

$A \rightarrow \varepsilon$

$B \rightarrow \varepsilon$

Step 01: First step for non-terminal

FIRST (A) = {ε}

FIRST (B) = {ε}

FIRST (S) = {a, b}

Not LL(1)

4A) Grammar

$E \rightarrow 2E_2$

$E \rightarrow 3E_3$

$E \rightarrow 4$

Passing input string 32423

Step 1: Initialization

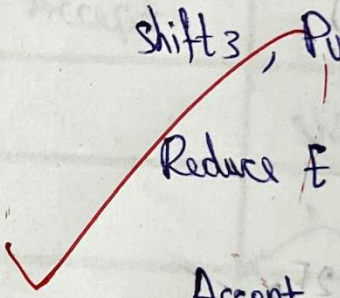
Stack = $

Input: 3 2 4 2 3 $

step 2: Parsing

S.No:25
P. Sathwik

Stack		
$	32423 $	Shift 3, Push 3
$3	2423 $	Shift 2, Push 2
$32	423$	Reduce ⇒ 4
		POP 4
$E	4 23$	Shift 4, Push 4
$E4	23$	Shift 2, Push 2
$E42	3$	Reduce E→4
		POP 4
$E4	3$	
		Reduce E→2E2
		POP E
$E	3$	Shift 3, Push 3
$E3	$	Shift 3, Push 3
$E3$	—	Reduce E→3E3
$E	—	Accept



Analytical Day-3

S.No: 25
P. Sathwik

① Syntax Directed translation

2 * (4+5)

E → E+T | E-+|T

T → T * F | T | F | F

F → (E) | num

E → Represent expression

T → Term

F → Factor.

SDT Action:

E → E+T { right = Pop(); left Pop(): exit

(left + right '+(+'); Push (left + Right +(+): }

E-T { Right = pop: left = Pop; Exist

left + Right + '-'); Push (left + right + '-');

}

T { result = Pop(); Push (result); }

T → T*F { right : Pop: left = Pop; exit left + right + '*' };

Push (left + Right + ('*'); }

F { result = pop(); push (result), }

F → (E) { result = POP(); Push (result); }

/num { Exist (num); Push(num); }

SNo: 25

P. Sathwik

(2) SSD Scheme

3*5+6*3

E → E+T / E→T / T

T → T*F / T / F / f

F → (E) / num

Semantic rule.

1. E → E1 + T {E.val = E1.val + T.val}

/E1 − T {E.val = E1.val − T.val}

/T. {E.val = T.val}

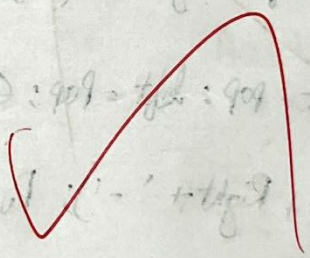
2. F → (E) {E.val = E.val

/num {E.val = num.val}

(3) Grammar G

E' → E

E → E+n / n



Parsing Action

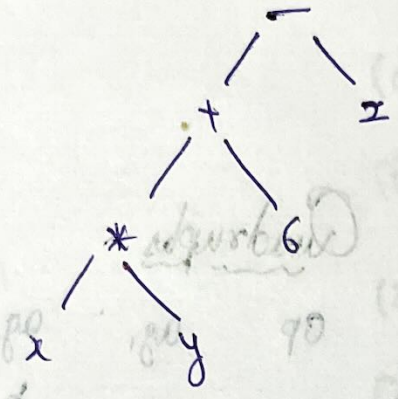
Stack	Input	Action
$	n+n$	shift n
$n	+n$	shift '+'
$n+	n$	shift 'n'
$n+n	$	E → n

$E	$	Reduce E → E+n
$E,	$	Reduce E' → E

S.No: 25
P. Sathwik

(4) Syntax Tree:

x * y + 6 - 2



(5) Syntax directed

S → EN

E → E+T / E-T / T

T → T*F / T/F / F

F → (E) / digit

input

5* 3+2

26/7/23

S.No: 25
P. Sathwik

Analytical Day-4 :-

Translate the expression $-(a+b)*(c+d)+(a+b+c)$ into

(i) quadruples

(ii) Triples

(iii) Indirect triples.

Given $-(a+b)*(c+d)+(a+b+c)$

$t_1 = a+b$

$t_2 = 0 - t_1$

$t_3 = c+d$

$t_4 = t_2 * t_3$

$t_5 = t_1 + c$

$t_6 = t_4 + t_5$

Quadruples

	op	ag_1	ag_2	Result
(0)	+	a	b	t_1
(1)	-	0	t_1	t_2
(2)	+	c	d	t_3
(3)	*	t_2	t_3	t_4
(4)	+	t_1	c	t_5
(5)	+	t_4	t_5	t_6

Triples :-

	op	arg_1	arg_2
(0)	+	a	b
(1)	-	(0)	-
(2)	+	c	d
(3)	*	(0)	(2)
(4)	+	(0)	c
(5)	+	(3)	(4)

Indirect triple :-

100 (0)

101 (1)

102 (2)

103 (3)

104 (4)

105 (5)

(2) Translate the expression $a^* - (b+c)$

S.No: 25

P. Sathwik

a) Quadruples

b) Postfix Notation

c) Three - address code.

Quadruples:-

$a^* - (b+c)$

$t_1 = b+c$

$t_2 = 0+1$

$t_3 = 0+a$

$t_4 = t_3 * t_2$

	op	arg1	arg2	result
(0)	+	b	c	t_1
(1)	−	0	1	t_2
(2)	+	0	a	t_3
(3)	*	t_3	t_2	t_4

b) Postfix Notation:

$a^* - (b+c) \Rightarrow abc + - *$

c) Three Address Code:-

$t_1 = b+c$

$t_2 = -t_1$

$t_3 = a^* t_2$

S.No:25
P. Sathwik

③
$t_1 = c * d$

$t_2 = a + b$

$q = t_2 - t_1$

④
$t_1 = a < b$

$t_2 = c < d$

$t_3 = d < c$

$t_4 = t_1 \text{ or } t_2$

$t_5 = t_4 \text{ and } t_3$

Back patching:-

if t_4 goto 4

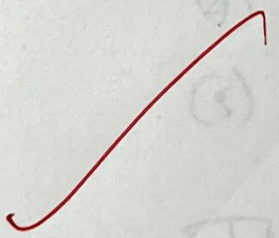
if not t_5 goto t_2

L_1:

goto t_3:

t_2

L_3: 1



27/7/23

Analytical Day-5

S.No: 25

P. Sathwik

(1) DAG for following three address Code:

$a = b + c$

$t_1 = a \times a$

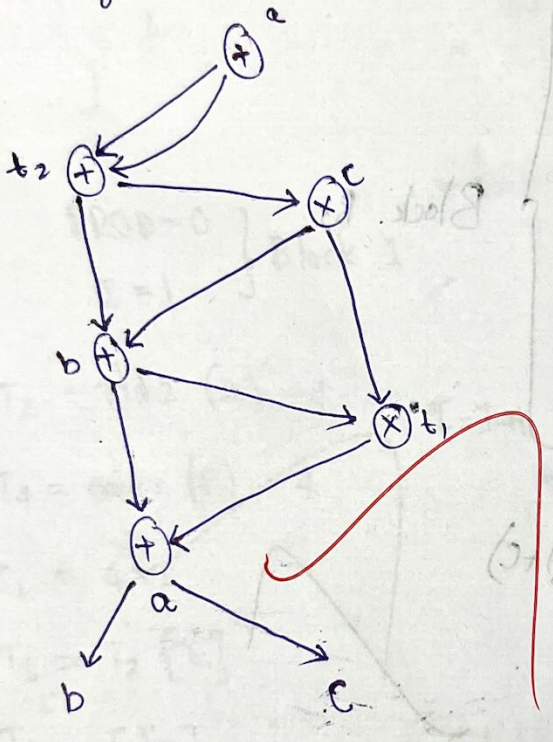
$b = t_1 + a$

$c = t_1 \times b$

$t_2 = c + b$

$a = t_3 + t_2$

(A) Directed cyclic graph:



(2) Basic blocks

(1) PROD = 0

(2) I = 1

(3) T_2 = addr (A) − 4

(4) T_4 = adr (B) − 4

(5) $T_1 = 4 \times 1$

(6) $T_3 = T_2 (T_1)$

(7) $T_B = T_3 \times T_5$

(8) $T_6 = T_3 \times T_5$

(9) PROD = PROD + T_6

(10) I = I + 1

(11) If ≤ 20 GoTo (3)

S.Nabi25
P. Sathwik

(A) :- PROD = 0

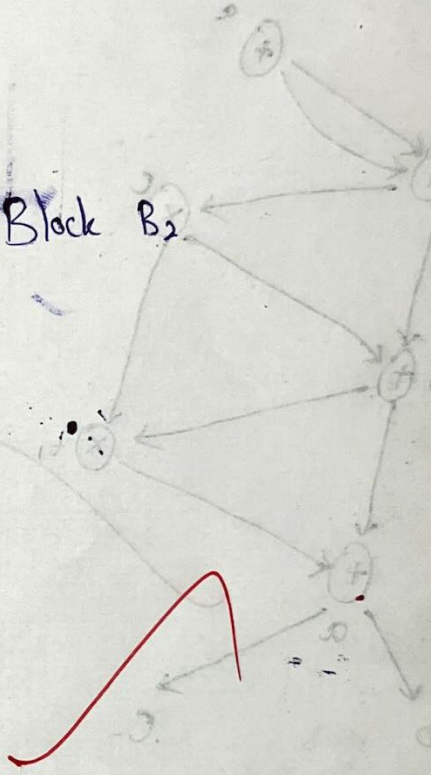
$T_1 = 4 \times I$

(1) PROD = 0
(2) I = 1
(3) T_2 = addr (A) − 4
(4) T_4 = addr (B) − 4

} Block B

(5) $T_1 = 4 \times I$
(6) $T_3 = T_2[T_1]$
(7) $T_5 = T_4[T_1]$
(8) $T_6 = T_3 \times T_5$
(9) PROD = PROD + T_6
(10) I = I + 1
(11) If I <= 6 GoTo (5)

} Block B_2



(3) W = (A+B) + (A+C) + (A+C)

t_1 = A+C

t_2 = A+C

t_3 = A+B

$t_4 = t_1 + t_2 + t_3$

W = t_4

Ans:- Code

```
MOV   A, R0
ADD   B, R0
ADD   A, R0
ADD   C, R0
ADD   A, R0
```

ADD C, R0

MOV R0, w

S.No: 25

P. Sathwik

3) Basic block and flow graph

```
for (i=1 to n)
{
    s=1;
    while (j<=n)
    {
        A=B*(C/D);
        j=j+1;
    }
}
```

If I <= 20 GoTo (2) } Block 3

Ans:

PROD = 0
I = 1
} Block 1

T_2 = addr (A) - 4

T_4 = addr (B) - 4

T_1 = 4 * I

T_3 = T_2 [T_1]

T_5 = T_4 [T_1]

T_6 = T_3 * T_5

PROD = PROD + T_6

I = I + 1

Block - 2

27/7/23

Analytical Day-6

28/07/23

i) Consider the following grammar and eliminate left recursion.

SNo: 25
P. Sathwik

$A \rightarrow ABd \mid Aa \mid a$

$B \rightarrow Be \mid b$

Step①: Identify left-recursion variables:

The left-recursive variables in the grammar are A & B, as they appear as the leftmost symbol in some of their production rules

Step②: Eliminate left recursion:

$A \rightarrow aA'$

$A' \rightarrow BaA' \mid \varepsilon$

$B \rightarrow bB'$

$B' \rightarrow eB' \mid \varepsilon$

(ii) Consider the following grammar & eliminate left recursion

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

Ans: Left recursion is eliminated by converting the grammar into a right recursive grammar.

$A \rightarrow \beta A'$

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

Explanation:

For non-terminal A, we create a new non-terminal A' to handle the recursion. Now, A can produce strings starting with β, A' can produce strings starting with Aα or ε. This way, we avoid left recursion for A.

S.No: 25
P. Sathwik

(iii) Do left factoring in the following grammar

a. S → bSSaS | bSSaSb | bSb | a

b. S → aSSbS | aSaSb | abb | b

a) Step①: Identify common prefixes

Common prefix in this grammar is "bSS"

Step②: Perform left-factoring:

factor out common prefix "bSS" & create new non-terminals to handle the different suffixes:

S → bSSS'

S' → aaS | aSb | b

b) ① Common prefix is "aS"

S → aSS'S'

S' → bS | aSb | ε

iv) Check whether the following grammar is a LL(1) or not

S.No: 25
P. Sathwik

$S \rightarrow iEtS \mid iEtSeS \mid a$

$E \rightarrow b$

Ans:

FIRST (S) = {i, a}

FIRST (E) = {b}

Follow sets:

FOLLOW (S) = {$}

FOLLOW (E) = {i, e, $}

28/7/25

Analytical-07

(1) Can truct a recursive discat parse for the following grammar using

S.No: 25
P. Jathwik

$E \rightarrow TE'$

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

$T \rightarrow fT'$

$T' \rightarrow +fT' / \varepsilon$

$f \rightarrow (E) / id$

```
Procedure E()
{
    T();
    E'();
}
```

$E \rightarrow TE'$

```
Procedure E'()
{
  if input symbol = '+' then
  advance();
  T();
  E();
}
```

$E \rightarrow +TE'$

```
Procedure T()
{
    F();
    T'();
}
```

$T \rightarrow +T'$

S.No:25
P. Sathwik

```
Procedure T()
{
  if input symbol / ("*' then
  advance();
  F();
  T'();
}
```

$T' \rightarrow *FT'$

```
Procedure F()
{
  if input symbol = 'id' then
  advance();
  else if input-symbol='(' then
  advance();
  E();
  if input symbol= ')'
  advance();
  else error();
  else error();
}
```

$F \rightarrow id$

$F \rightarrow (E)$

(2) Construct the follows grammar for operator procedure parser.

Sol:

	a	(	)	,	$
a		>	>	>	>
(	<	>	>	>	>
)	<	>	>	>	>
,	<	<	>	>	>
$	<	<	<	<	

S.No: 25

P. Sathwik

Step-1

$\$(a,(a,a))\$$

We insert precedence operator b/w symbol

$\$ < (< a > , < (< a > , < a >) >) > \$$

Step-2:

We can scan and parse the string as

$\$ < (< a > , < (< a > , < a >) >) > \$$

$\$ < (S_1 < (< a > , < a >) >) > \$$

$\$ < (S_1 < (S_1 < a >) >) > \$$

$\$ < (S_1 < (S_1 S) >) \$$

$\$ < (S_1 < (L , S) >) > \$$

$\$ < (S_1 < (L) >) > \$$

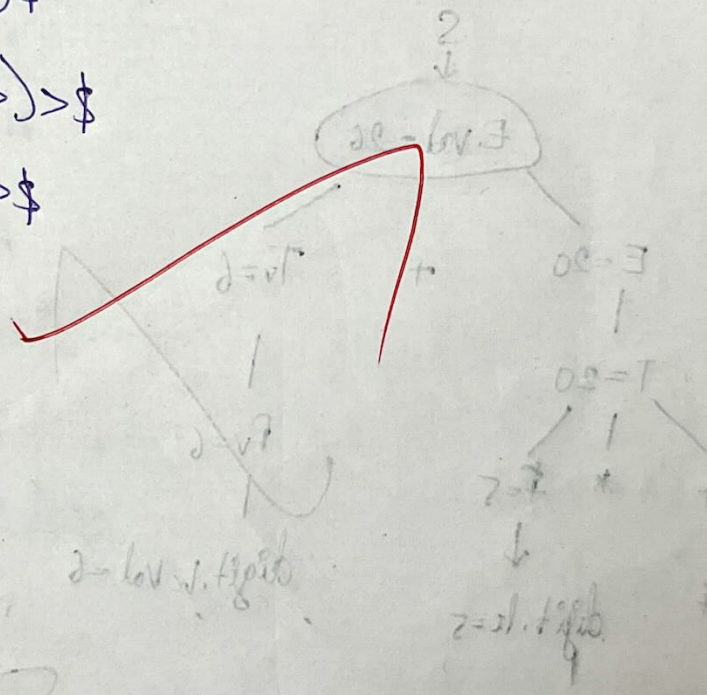
$\$ < (S_1 S) > \$$

$\$ < (L , S) > \$$

$\$ < (L) > \$$

$\$ < S > \$$

$\$ \$$

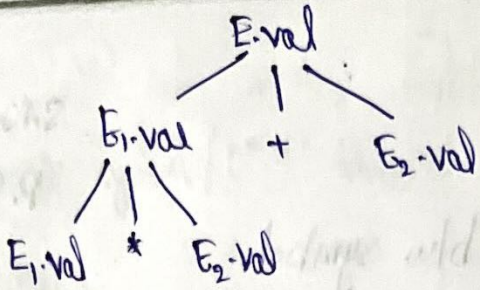


(3) Design the dependency graph for the following grammar.

$E \rightarrow (E_1 + E_2$

$E \rightarrow E_1 * E_2$

Production	Semantic Rules
$E \rightarrow E_1 + E_2$	$E.val \rightarrow E_1.val + E_2.val$
$E \rightarrow E_1 * E_2$	$E.val \rightarrow E_1.val + E_2.val$



S.No:25
P.Sathwik

(4) Design Rw dependency graph for the following grammar.

S → T List

T → int

T → float

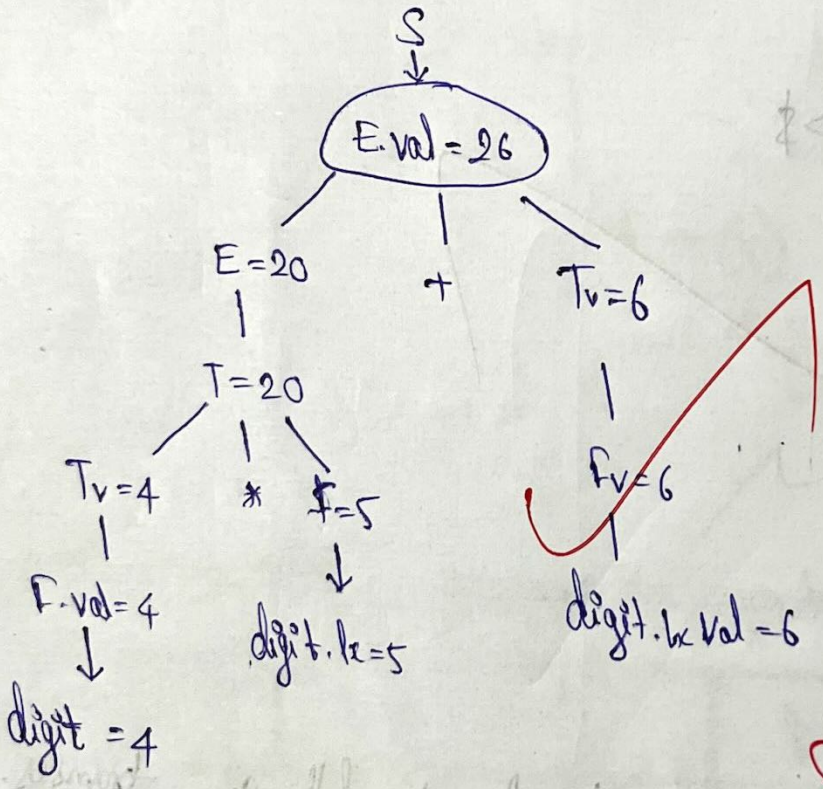
T → class

T → double

List → List, id

List → id

Design for above grammer of dependency graph.



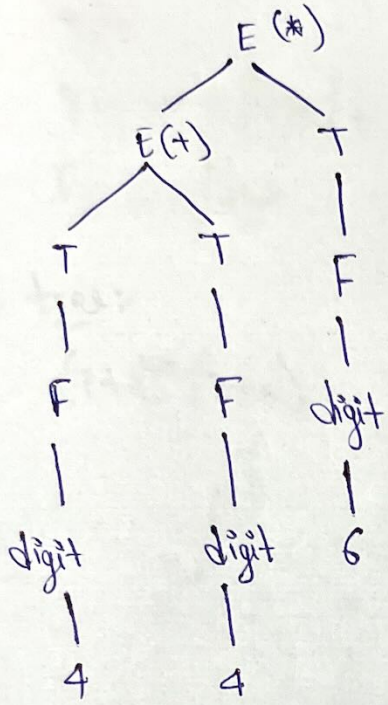
Analytical Day-08.

S.No: 25

P. Sathwik

(1) SDD for simple desk. Parse tree for expression $(6+4)*(4+3)n$.

A Parse Tree:-



SDD:-

$S \rightarrow E$ id

$E \rightarrow E+T \mid E-T \mid T$

$T \rightarrow T*F \mid T/F \mid F$

$f \rightarrow (E) \mid$ digit

$S \rightarrow E$ { n.val = E.val }

$E \rightarrow E+T$ { E.val = E.val + T.val }

$E \rightarrow E-T$ { E.val = E.val - T.val }

$E \rightarrow T$ { E.val - T.val }

$T \rightarrow T*F$ { T.val = T.val * F.val }

$T \rightarrow T/F$ { T.val = T.val / F.val }

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

F → (E) { F.val = E.val }

F → digit { F.val = digit.val }

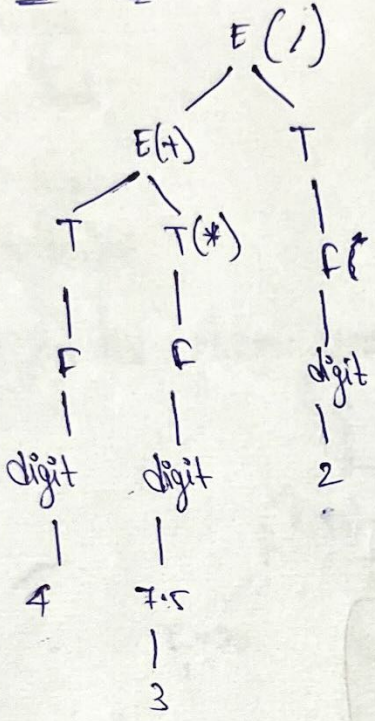
S.No:25

P. Sathwik

(2) Parse Tree according to syntax directed.

(4 + 7.5 * 3) / 2

A. Parse Tree:-



Evaluate (7.5 * 3) = 22.5

Evaluate (4 + 22.5) = 26.5

Evaluate (26.5 / 2) = 13.25

result = 13.25

(3) SDD for operators + and * variable

X * (3 * X + X + X)

S → E

E → E + T

E → T

T → T + F

F → X

(E)

IF

S.No:-25
P. Vathsavi

① SDD for simple desk parse tree (7+4)* (3+6)n.

S → En

E → E+T / E

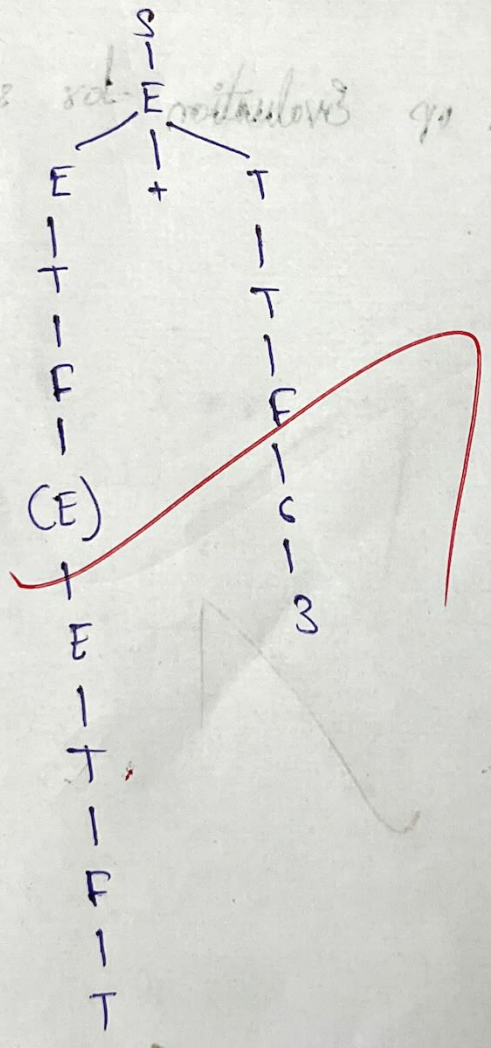
T / T

T → T* F / T / F

F → (E) / digit

A) Parse tree:-

(7+4)* (3+6)



S → E {n.val = E.val}

E → E₁ {E.val = E₁.val} +T {E.val = E.val + T.val} / T {E.val = T.val}

T → T₁ {T.val = T.val}* F {T.val = T₁.val + F.val} / F {T.val = F.val}

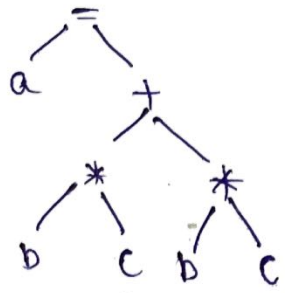
F → (E {E.val = E.val}) | digit {F.val = digit.val}

S.No: 25
P. Bathwal

(2) Construct the syntax tree for expression

$a = b * c + b * c$

A:- $a = b * c + b * c$



(3) Illustrate the bottom up Evaluation for attributes.

5 * 6 + 4

S → EN

E → E + T

E → E - T

E → T

T → T * F

T → T/F

T → F

F → E

E → digits.

Grammar:

S → EN

Input : 5 * 6 + 4

S.No: 25
P. Sathwik

Step 1: Shift the digit in stack.

stack: [5]

Input: *6+4

Step 2: Shift the * in stack

stack: [5, *]

Input: 6+4

Step ③: Shift the 6 number in stack

stack: [5, *, f] T → F (F → E → dig)

Input: +4

Step 4: Redu E → T

Stack: [E]

Input: +4

Step 5: [E, +]

4

Step 6: F → E → digit

[E, +, F]

Step 7: Redu T → F

[E+, f)

(4) SDD scheme that translates

5*7

8*4

S.No: 25

P. Jothwik

A Grammar:-

1. Input 5*7

S.inh = E.val

E1.inh = T.val

T1.inh = F.val

T2.inh = F.val

F1.val = 5, F1.code = 5

F1.val = 7, F2.code = 7

T2.val = 7, T2.code = 7

T1.val = 5, T1.code = 5

E1.val = 5, E1.code = 5

E.val = 5, E.code = 5

o/p = 57*.

2. Input 8 + 4

S.inh = E.val

E1.inh = T.val

T1.inh = F1.val

T2.inh = F2.val

F1.val = 6, F1.code = 8

F2.val = 4, F2.code = 4

T2.val = 4, T2.code = 8

F1.val = 8,

E.val = 8, E.code = 8

o/p : 84*

Analytical Day-10

S.No: 25
D. Sathwik

(1) Quadraples for expression $(a+b)+(c+d)-(a+b/c+d)$

A Let,

$t_1 = a+b$

$t_2 = -t_1$

$t_3 = c+d$

$t_4 = t_2 * t_3$

$t_5 = t_1 + c$

$t_6 = t_4 - t_5$

Quadruple:

	Operator	Arg1	Arg2	Result
(0)	+	a	b	t_1
1	−	t_1		t_2
2	+	c	d	t_3
3	+	t_2	t_3	t_4
4	+	t_1	c	t_5
5	−	t_4	t_5	t_6

(2) Construct the address code following executed if A<B and C<D then t=1 else t=0

A If A<B and C<D

t = 1

else

t=0;

These address code

S.No: 25
P. Sathwik

(1) if (A<B) goto 3

(2) goto (4)

3) if (C<0) goto 6

4) t = 0

5) goto (7)

6) t=1

(3) Generator the address code.

```
C=0
do
{
    if (a<b) then
    x++;
    else
     x--;
     c++;
}while (c<5)
```

A:- There address code

① c=0

② if(a<b) goto(4)

③ goto (4)

④ T_1 (x+5)

⑤ X = T_1

⑥ goto (9)

⑦ T_2 = X-1

⑧ X = 12

⑨ T_8 = C+1

⑩ C- T_3

⑪ if (C<5)

⑫ Goto (2)

S.No:25
P. Sathwik

(4) Generate three address code

```
int a(40), b[10], i, dp=0;
for (i=0; i<10; i++)
{
    dp += a[i] + b[i];
}
```

(A) (1) $i = 0$

(2) $t_1 = i < 10$

(3) if not t_1 goto 9

4. $t_2 = i + 4$

5. $t_3 = a[t_2]$

6. $t_4 = i + 4$

7. $t_5 = b[t_4]$

8. $t_6 = t_3 + t_5$

9. $t_7 = dp + t_6$

10. $dp = t_7$

11. $i = i + 1$

12. goto 2.