20/01/23

Analytical Question (Topic-1).

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

```
int main()
{
int a=10,b=30;
if (a<b)
 return (b);
Else
return (a)
}
```

Sol:- To find the no of tokens in the given C statement.

The breakdown of the code into tokens;

1. int : keyword
2. main : Identifier
3. (: left parenthesis
4.): Right parenthesis
5. {: left curly brace
6. /* ... */ → comment
7. int
8. a
9. 10
10. ;
12. b (Variable name)
13. = Assignment
14. 30.
15. ;
16. if

17. C;
18: a
19: <: less than
20 b: (Identifier)
21);
22. return
23. C:
24 b
25)
26 ;:
27 else
28 return
29 C:
30 a:
31):
32: ;:
33. }:

∴ Total no of tokens = 33

2. Write a regular Expression to denote set of all strings over {0,1}* containing substring 101.

Sol (0+1)* 101 (0+1)*

The regular Expression (0+1)* 101 (0+1)* will match any string that contains the substring "101" as a contiguous sequence of characters.

3. Write a regular Expression that have atleast two consecutive 0's (0,) (1's

A. (00+|11+)

| → act as OR Operator

00+ least two consecutive 0's

11+ " 1's

for Example this regular Expression will match strings like "0001", "1100", "00", "1111", "000000", "111111", "0101" etc.

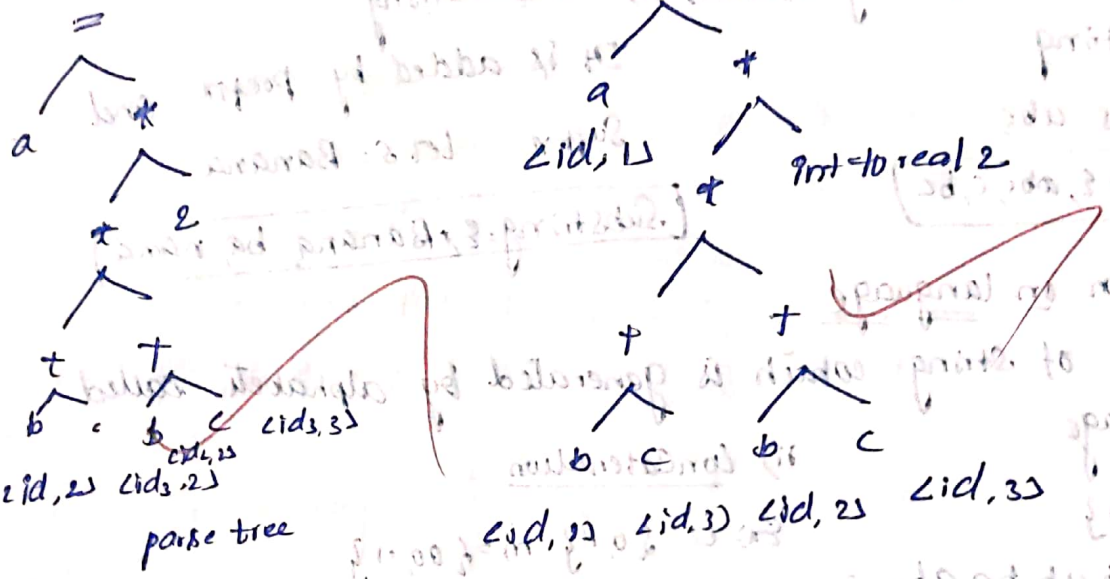
2:

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

1-Lexical Analysis:-

<a> <=> <c> <b> <+> <c> <*> <c> <b> <+> <c> <*> <2>

2. Syntax Analysis



3. Semantic Analyzer.

ICG

```
temp1 = id2 + id3
temp2 = temp1 * temp1
temp3 = int to real 2
temp4 = temp2 * temp3
id1   = temp4
```

Code Optimizer

```
temp1 = id2 + id2
temp2 = temp1 * temp1
id1   = temp2 * 2.0
```

Code Generator

```
MOV f   id2, R1
MOV f   id3, R2
ADD f   R1, R2
MUL f   R2, R2
MOV f   #2.0, R3
MUL f   R2, R3
MOV     R3, id1
```

21/07/23 Operation of string

P. Vishnusai 8

1) length of string 2. Empty string 3. prefix of string

Ex: $S = 10101$ Ex: $S = 0$ Any no of leading symbols of string let $s = abc$

$\therefore |S| = 5$ $\therefore |\varepsilon| = 0$ Prefix = ε, abc, a, ab

4. Suffix of string

Any no of trailing symbols in the string

let $s = abc$

Suffix = ε, abc, c, bc

5. Sub string

ε is added by prefix and Suffix let s = Banana

Substring = ε, Banana, ba, nana

Operation on language

⇒ A set of string which is generated by alphabets called Language

$\Sigma = \{a, b\}$

$L = \{a, b, ab, ba, aa, bb \ldots\}$

i) Union

Ex $L = \{0, 1\}$ $m = \{00, 11\}$

$\therefore L \cup m = \{0, 1, 00, 11\}$

ii) Concatenation

Ex: $L = \{0, 1\}$ $m = \{00, 11\}$

$\therefore Lm = \{000, 011, 100, 111\}$

iii) Kleene closure : (L^*)

Ex $\Sigma = \{a\}$ $\therefore \Sigma^* = \varepsilon, a, aa$

iv) positive closure (L^+) Ex: $\Sigma = \{a\}$ $\therefore \Sigma^+ = a, aa$

1. find out how many tokens are there in the regular Expression

a) printf("%d", &x); → 7 token

b) scanf("%d", &x); → 8 token

c)
```
main()
{ int a, b, c;
  clan();
}
class
{ while<(1)
  { printf("%d", c);
  }
```

→ 3 tokens

B → ε | {ε} | {b, q}

(LC1) Grammer

1) Construct the predictive parser for the following grammar.

S → a | ↑ | (T) T → T, S | S

Sol S → a | ↑ | (T) T → S | S

i) S → a | ↑ | (T)

T → T, S | S

Step1:- Eliminate left recursion from grammer

S → a | ↑ | (T)

T → ST'

T' → S | ε

Step2:- Create the first & follow sets for each non-terminal

Set-1

first(S) = {a, ↑, (}

first(T) = {a, ↑, (}

Set-2

Follow(S) = {$,)}

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

Step3:- Predictive parsing table

	a	↑	(	)	$
S	a	↑	(		

parsing table for T

	a	↑	(	)	,	$
T	a	↑	(			
T'					,	

2a) S → CC

S → CCC → CCId

Step1:- Argment the grammer

S' → S

S → CC

C → C | d

Step2:- Computer the closure and go to sets for item.

LR(0) items

1. S' → S
2. S → .CC
3. S → .CC
4. S → d
5. C → CC
6. C → d

10

Closure (I0)

1. S' → S

Go To(I1);

Go To (I1, C) = I2

12 closure (I2)

1. S → CC

2. S → CC

3. S → d

4. C → CC

7. C → d

Go To(I2):

Go To (I2, C) = I3

Go To (I2, C) = I4

Go To (I2, d) = I5

13.

Closure (I3)

1. S → C.C

2. C → .CC

3. C → .d

Go To(I3):

Go To(I3, C) = I8

14. closure (I4)

S → C.C

C → C.C

C → .d

Go To(I4)

Go To (I4, C) = I7

15. Closure (I5)

12. Closure (I7)

16.

Closure (I6):

1. C → C.C

2. C → .CC

3. C → .d

Go To (I6, C) = I9

18. Closure (I1)

1. C → d

	c	d	$	S	C
I0				Shift 1	
I1			Accept		
I2	Shift 3	Shift 4		Shift 6	Shift 6
I3	Shift 3	Shift 4		8	9
I4	11	11			
I5			Reduce 2		

(b) grammar

S → 2B2

B → 3B3

B → 4

input string 32423

Step1 Initialization

stack : $

input 32423$

Step2 parsing

stack	input	Action
$	32423$	Shift 3, push 3
$3	2423$	Shift 2, push 2
$32	423$	Reduce
$ 6	423$	pop
$ ∈ 4	23$	Shift, push 4
$ ∈ 42	3$	Shift 2 push 2
$ ∈ 4	1	Pop 4.
$ ∈	2$	Shift, p
$ ∈ 3	$	Shift, push
$ ∈ 3$	-	Pop ∈ → 3$
$ ∈	-	Accept

12/12/23

F → id

follow(E) = {$,), +}

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

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

10/2/23

Analytical Day-3

1. Syntax Directed Translation 2*(4+5) Context free grammar for arithemetic expression

Sol E → E+T | E-T | 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(); Emit (left + right) + ('+');
push (left + right + '+');}
|E-T {right = pop(); left = pop(); Emit (left + right + '-'); push (left +
right + '-');}
|T {result = pop(); push (result);}
T → T*F {right = pop(); left = pop(); Emit (left + right + '*')
push (left + right + ('*');}
|T / F {right = pop(); left = pop(); Emit (left + right '/'); push
(left + right + ('/');}
|F {result = pop(); push (result)}
|num {Emit (num); push (num);}

2) GSD Scheme — Semantic Rule

3*5+6*3

E → E+T | E-T | T
T → T*F | T/F | F
F → (E) | num

1. E → E_1+T {E.val = E_1.val + T.val}
|E_1 - T {E.val = E_1.val - T.val}
|T {E.val = T.val}
2. T → T_1*F {T.val = T_1.val + F.val}
|T_1 / F {T.val = T_1.val - F.val}
|F {T.val = F.val}
3. F → (E) {F.val = E.val}
|num {E.val = num.val}

3) Grammar G

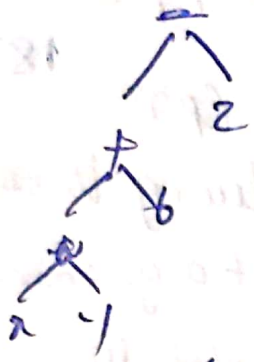
E' = E
E → E+n|n

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

4) Syntax tree for expn

a) $x^{\wedge}y + 6 - 2$



(1) Syntax-directed

$s \to E$

$E \to E+T \mid E-T \mid T$

$T \to T^{\wedge} F \mid T \mid F \mid 12$

$F \to (E) \mid digit$

$\alpha \to ;$

17

1. Attributed Grammar

$S \to E \{ S.val = E.val \}$

$E \to E_1 + T \{ E.val = E_1.val + T.val \}$

$\mid E_1 - T \{ E.val = E_1.val - T.val \}$

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

$T \to T_1 \wedge F \{ T.val = T_1.val \wedge E.val \}$

$\mid T_1 \mid F \{ T.val = T_1.val \mid E.val \}$

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

$F \to (E) \{ F.val = E.val \}$

$digit \{ E.val = digit.val \}$

$\alpha_1 \to \{ \alpha_1.val = ; \}$

→ result = $5^{\wedge}3+2$

$5^{\wedge}3 = 125$

$125 + 2 = 127$

$5^{\wedge}3+2 = 127$ //.

24/07/23 Operator Precedence parsing

Properties

Analytical Topic –

1. Translate the Expression -(a+b) * (c+d) + (a+b+c) into

i) Quadruples

ii) Triples

iii) Indirect triples.

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

$t_1 = a+b$

$t_2 = -t_1$

$t_3 = c+d$

$t_4 = t_2 * t_3$

$t_5 = t_1 + c$

$t_6 = t_4 + t_5$

Quadruples

	op	arg1	arg2	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

Triple

	op	arg1	arg2
(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)

3) Translate the expression a*-(b+c)

a) Quadruples

b) postfix notation

c) Three-address code

(a) Quadruples

a*-(b+c)

$t_1 = b+c$

$t_2 = 0 - t_1$

$t_3 = 0 + a$

$t_4 = t_3 * t_2$

	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) ⇒ abc+-*

c) Three-address code

$t_1 = b+c$

$t_2 = -t_1$

$t_3 = a * t_2$

4) Write down translation scheme to generate three address code for

g := a+b-c*d

$t_1 = c*d$

$t_2 = a+b$

$g = t_2 - t_1$

4) Translate (a<b) or (c<d) and (d<e) into three address statement using back patching.

sol

$t_1 = a<b$

$t_2 = c<d$

$t_3 = d<e$

$t_4 = t_1$ or t_2

$t_5 = t_4$ and t_3

Back patching

if t_4 goto 4

if not t_5 goto L2

4:

goto L3:

L2

L3:

id := ma
i 19
id

28/11/23

Analytical Questions

2. compute the basic blocks for the given three address statements (using concept map)

(1) PROD = 0 ✓

(2) I = 1

(3) T2 = addr (A) - 4

(4) T4 = addr (B) - 4

(5) T1 = 4 × I ✓

(6) T3 = T2 [T1]

(7) PROD = PROD + T3 ✓

8. I = I + 1

9. If I <= 20 goto (5)

10. j = j + 1 ✓

11. k = k + 1

12. if j <= 5 goto (7)

13. i = i + d

Sol To determine basic blocks.

1. PROD = 0

So After conditional also leader.

leader.
1
5
7
10
13

To determine basic blocks

Start → leader → Before the next leader

```
PROD = 0
I = 1
T2 = addr(A) - 4
T4 = addr(B) - 4
```
B1

```
T1 = 4 * I
T3 = T2[T1]
```
B2

```
PROD = PROD + T3
I = I + 1
If I<=20 goto (5) B2
```
B3

```
J = J + 1
If J<=20 goto (7) B3
```
B4

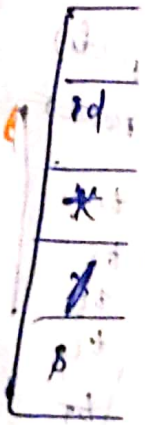
```
i = i + j
```
B5

5 6

7 9

B1, B2, B3, B4, B5

same flow graph



② Construct basic blocks & flow graph & identify loop invariant statements for (i=1 to n)

```
{ j=1;
while (j<=n)
{ A = B*(C+D);
j = j+1
}}.
```

Given

Stack

$

$ id

$ E

Basic Blocks

1) i = 1 to n

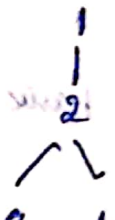
j = 1

While (j<=n)

T1 = B*(C+D)

T2 = j = j + 1

Flow Graph :-



Loop invariant statements.

⇒ The value of i stays constant in each of the loop.

3. Generate assembly language for x=(a+b)*(c+d)-(a+c)

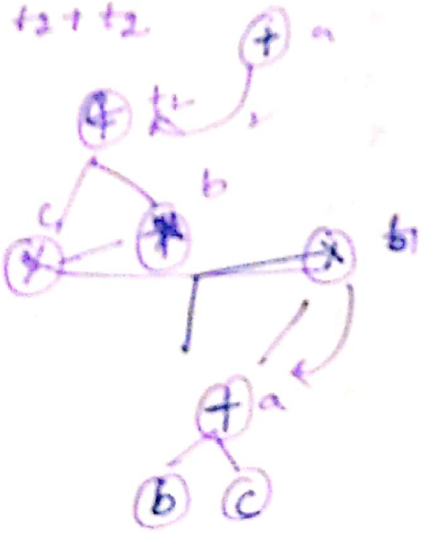
```
MOV A, R1
MOV B, R2
ADD B, R1
ADD C, R2
ADD R2, R3
ADD R1, R2
MOV R3, x
```

4. Construct a DAG for the following three address code.

Code

```
a = b+c
t1 = a*a
b = t1+a
c = t1*b
t2 = c+b
a = t2+t2
```

Sol



Analytical day-13

P. VISHNU SAI
192011177

1- Quadruples for Expressions

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

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

$t_1 = a+b$

$t_2 = -t_1$

$t_3 = c+d$

$t_4 = t_2 * t_3$

$t_5 = t_1 + c$

$t_6 = t_4 - t_5$

Quadruples :-

	Operator	arg1	a.	
(0)	+	a	b	t_1
(1)	−	t_1	d	t_2
(2)	+	c	t_3	t_3
(3)	*	t_2		t_4
(4)	+	t_1	c	t_5
(5)	−	t_4	t_5	t_6

2. Construct three Address code following Expressions.

if A<B and c<D then t=1 else t=0

A) if A<B & c<D

t = 1

Else

t = 0

Three address code

1) if (A<B) goto (3)
2) goto (4)
3) if (C<D) goto (6)
4) t=0
5) goto (7)
6) t=1
7) -

3 Generate three address code

c=0
d=0
{ if (a<b) then
x++;
Else
x--;
c++;
} while (c<7)

4 Three address code

if (i<5) goto (2)

i=0
if(a<b) goto(4)
goto(7)
$T_1 = x + 1$
x = T_1
goto(9)
$T_2 = x - 1$
x = T_2
T_3 = (1)

4. Generate three address code

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

1-qu

A)

1) i = 0
2) t1 = i < 10
3) if not t1 goto 9
4) t2 = i * 4
5) t3 = a[t2]
6) t4 = i * 4
7) t5 = b[t4]
8) t6 = t3 * t5
9) t7 = dp + t6
10) dp = t7
11) i = i + 1
12) goto 2

2.