Error Recovery schemes.

→ local correction checking

→ Global correction checkings.

## Analytical questions (Topic-1)

20/07/23

Find the no-of tokens in the following statement

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

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 Parantheris
4. ); Right Parantheris.
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 (;

18: a
19: <: less than
20: b: (Identifier
21: );
22: return
23: (;
24 b;

25 )   31 ?;

26 ;;   32 :1);

27 else.   33 };

28 return;

29 c

30 a;

∴ Total no. of tokens = 33.

2. write a regular expression to denote set of all strings over {0,1}* containing sub strings.

Sol:- $(0+1)^* \cdot 101 (0+1)^*$

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 (or) 1's.

Ans:- $(00+11+)$

| → act as OR operator

00 + least two consecutive 0's

11 + " " " 1's.

for Example this regular expression will match string.

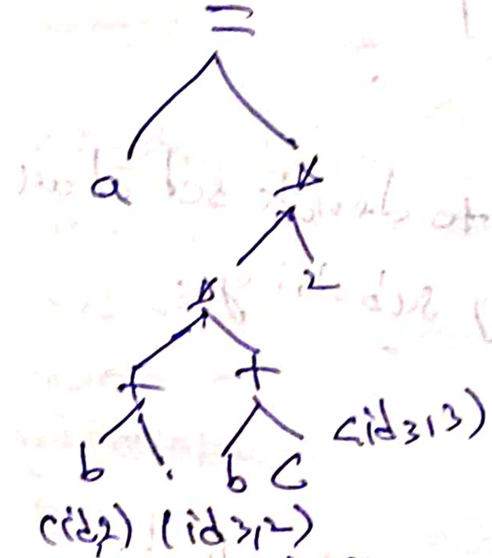
we "0001", "1100", "100", "1111", "000000", "11111", "0110"

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

1. Lexical Analysis :-

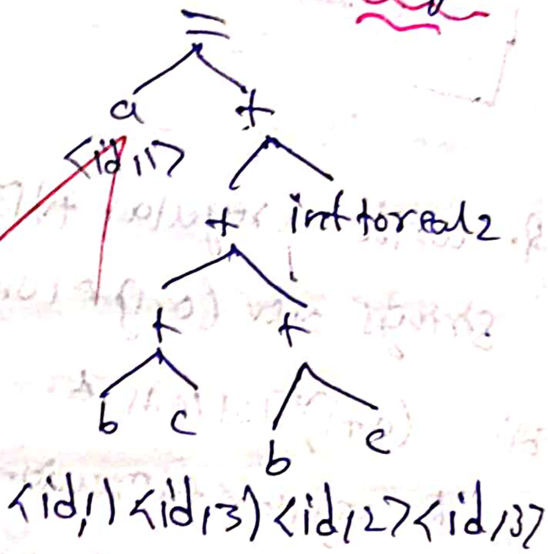
<a> <=> <(> <b> <+> <c> <*> <(> <b> <+> <c> <*> <2>

2. Syntax Analysis



Phase tree.

3. Semantic Analyzer.




ICG :-

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

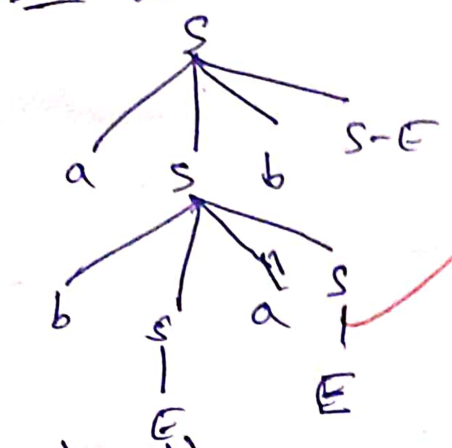
code optimizer.

temp1 := id2 + id3
temp2 := temp1 * temp1
id1 = temp2 * 2.0

code Generator.

mov + id2, R1
mov + id3, R2
ADD + R1, R2
mul + R2, R2
mov + #2.0 R3
mul + R2, R3
mov R3, id1

step:-2




(non-termial)
not considered.
only consider

1) Analytical question TOPIC:-2.

1) Determine whether the following grammer.

$S \to AaAb / BbBa$.

$A \to E$ is LL(1) or not?

$B \to E$

Sol:-

| | FIRST | Follow. |
|---|---|---|
| $S \to AaAb, BbBa$ | {a, b} | {S} |
| $A \to E$ | {E} | {a, b} |
| $B \to E$ | {E} | {b, a} |

| | a | b |
|---|---|---|
| S | $S \to AaAb$ | BbBa |
| A | $A \to E$ | $A \to E$ |
| B | $B \to E$ | $B \to E$ |

$\boxed{\text{LL(1) Grammer.}}$

1) Construct the Predictive parser for the following grammer.

$S \to a | T | (T) T \to T, S | S$.

Sol:- $S \to a | \uparrow | (T) T \to S | S$.

i) $S \to a | \uparrow | (T)$ $T \to T S | S$.

13

step 1:- Eliminate left recursion from grammer

$S \to a | T | (T)$

$T \to ST'$

$T' \to S | \epsilon$

step 2:- create the first & 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:

| | a | ↑ | ( | ) | , |
|---|---|---|---|---|---|
| S | a | ↑ | ( | | |

Parsing table for T:

| | a | ↑ | ( | ) | , | \$ |
|---|---|---|---|---|---|---|
| T | a | ↑ | ( | | | |
| T' | | | | | | |

24) SLR.

$S \to CC \to cC | d$

step 1:- Argment the grammer.

$S' \to S$

$S \to CC$

$C \to cC | d.$

step 2:- computer the closure and go sets for item LR(0) items

1. $S' \to S.$
2. $S \to CC.$
3. $S \to cC$ :
4. $S \to d.$
5. $C \to cC.$
6. $C \to d.$

I0.

closure (I0)

1. $S' \to S$

goTO (I1);

goTO (I1, C) = I2.

goTO (I2);

goTO (I2, 1) = I3

goTO (I2, c) = I4.

goTO (I2, d) = I5.

I3.

closure.(I3)

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

goTO (I3)

goTO (I3, c) = I8.

14.

14. Closure (14)

S → c.C

C → c.C

C → d

Go To (14)

Go To (14, c) = 17

15. Closure (15)

16:-

Closure (10)

1. C → c.C
2. C → CC
1. C → d
2. G(16, C) def.

15. Closure (11)

1. C → d.

| | c | d | d | S | C |
|---|---|---|---|---|---|
| 10 | | | | shift 1 | |
| 11 | | | Accept | | |
| 12 | SLt13 | shift | | shift | shift 6 |
| 13 | SLt13 | shift | | 11 | 11 |
| 14 | 11 | shift | | 11 | 11 |
| 15 | | | | | |
| 16 | | | Reduce | | |
| 17 | | | | | |
| 18 | | | | | |

(4) grammer

E → 2E2

E → 3E3

E → 4

input string 32423

Step 1:- Intilization.

stack : 1

Input 32423$

Step 2:- Parsing

| stack | input | Action |
|---|---|---|
| $ | 32423$ | Shift 3 Push |
| $3 | 2423$ | shift, Push |
| $32 | 423$ | Reduce |
| $6 | 423$ | Pop |
| $E4 | 23$ | shift, Push |
| $E42 | 3$ | shift 2 push 2 |
| $E4 | 4 | POP 4 |
| $E | 2$ | shift, Push 3 |
| $E3 | $ | shift push 3 |
| $E3$ | — | Red E → 3E3 |
| $E | — | Accept. |

22/7/23

| | | |
|---|---|---|
| 83 | 24238 | shift 2 |
| 832 | 4238 | shift 4. |
| 8324 | 238 | shift 2 |
| 832E | 238 | 2E2 → E |
| 832E2 | 38 | shift 3 |
| 83E | 38 | |
| 83E3 | 38 | 3E3 → E |
| 8E | 8 | Accept - |

| stack | input Buffer | Action |
|---|---|---|
| 8 | 85985 | shift 8 |
| 88 | 5958 | shift 5 |
| 885 | 4585 | shift 4 |
| 8854 | 58 | 4 → T |
| 885T | 585 | shift 5 |
| 8885 | 68 | STS → T |
| 88T | 85 | shift 8 |
| 8T8 | | |
| DE | 8 | 8T8 → E Accepted. |

## Analytical Topics :-

1. Translate the Expression −(a+b)*((c+d)+(a+b+c)) into
i) quadraple.
ii) triple.
iii) Indirect triple.

sol:- Given −(a+b)*((c+d)+(a+b+c))

t1 = t1 − t1
t3 = c+d.
t4 = t2+t3
t5 = t2+c.
t6 = t4+t5.

Quadruples.

| | op | arg1 | arg2 | Result |
|---|---|---|---|---|
| 0 | | | | |
| 1 | + | a | b | t1 |
| 2 | − | 0 | t1 | t2 |
| 3 | + | c | d | t3 |
| 4 | * | t2 | t3 | t4 |
| | + | t1 | c | t5 |
| 5 | + | t4 | t5 | t6. |

Triple.

| | op | arg1 | arg2 |
|---|---|---|---|
| (0) | + | a | b |
| (1) | + | (0) | − |
| (2) | | | |
| (3) | + | c | d. |
| (4) | * | 0 | 2 |
| (5) | + | (0) | c |
| | + | (3) | (4) |

Indirect triple :-

| | |
|---|---|
| 100 | (0) |
| 101 | (1) |
| 102 | (2) |
| 103 | (3) |
| 104 | (4) |
| 105 | (5) |

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

a) quadruples.

b) post fix notation.

c) three address code.

a) Quadruples.

$a^* - (b+c)$.

$t1 = b+c$.

$t2 = 0+1$

$t3 = 0+a$.

$t4 = t3 * t2$.

| | op | arg1 | arg2 | result |
|---|---|---|---|---|
| (0) | + | b | c | t1 |
| (1) | − | 0 | 1 | t2 |
| (2) | + | 0 | a | t3 |
| (3) | * | t3 | t2 | t4. |

b) post fix notation:- $a^* - (b+c) \Rightarrow abc+-*$

c) Three address code:

$t1 = b+c$.

$t2 = -t1$

$t3 = a^* t2$.

(4) write down translation scheme. generate three address code.

$g = a+b-c*d$.

$t1 = c*d$.

$t2 = a+b$.

$g = t2 - t1$

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

sol:- $t1 = a<b$.

$t2 = c<d$.

$t3 = d<e$.

$t4 = t1 \text{ or } t2$

$t5 = t4 \text{ and } t3$

Back patching

if t4 goto L1.

if not t5 goto L2

L1:

go to L3.

L2

L3 1.

25/7/25

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 = 4addr(B) - 4
5) T1 = 4 x 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 + j.

Sol:- To determine basic blocks.

1) PROD = 0.

After conditional also - leads:-

leader.

1

5

7

10

13.

To determine basic blocks:

starts -> leader -> Before the next leader

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

T1 = 4 x I
T3 = T2(T1) B2.

7 --- 9 B3.

PROD = PROD + T3.
I = I + 1
If I <= 20 go to (5) B2

j = j + 1
If j <= 5 goto (7) B3

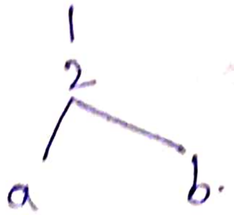
B4 . 10 --- 12

i = i + j B5 . 13.

(5) co
inv
B
1)
whi
A =
3) c
mo
AD
AD
AD
AD
mo
Sol:-

(2) construct basic blocks & flow graph & identify loop invariant statement for (i=1 to n).

flowgraph:-




{ i=1;
while (i<=n)
{
A=B(C/D)
i=i+1
}
}

S= i:=i+1.

loop invariant statement:-

=> The value of i stays constant in each of the loop.

Basic Blocks:-

1) i=1 to n.
i=1
while (i<=n)
A=B*(C/n).

3) Generate assembly languages for N=(A+B)+(A+C)+(A+C)

MOV A,R1
MOV R1,R3
ADD B,R1
ADD C,R3
ADD R3,R3
ADD R1,R3.
MOV R3,N.

2) Construct a DAG for the following three add code.

a=b+c;
t1=a+a.
b=t1+a.
c=t1*b.
t2=c+b.
a=t2+t2.

sol:-