21-02-24
Wednesday

Compiler Design

Day-1 Analytical

① (a) Find the number 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);
}
```

(b) Find the number of tokens in the following C Statement is:-

```
main()
{
a = b+++---+++==;
Printf ("%d%d", a, b);
}
```

(a)
- int - a - 30 - b - ; - ;
- main - = - ; -) - else - }
- (- 10 - if - return + return
-) - , - (- (- (
- { - b - a - b - a
- int - = - < - b -)

(b)
- main - = - (- b
- (- b - "%d %d" -)
-) - = - , - ;
- { - =; - a - }
- a - Printf - ,

Page
Date

② @ Find the number of tokens in the following C Statement is:

```
main()
{
int a=10;
Char b="abc";
int c=30;
Char d="xyz";
in /*Comment*/t m=40.5;
}
```

(b) Identify the lexical errors in the following C Statement is:

```
Void main()
{
int x=10, Y=20;
Char *a;
a=&x;
x=1xabi;
}
```

@ main() ——> 3
{ ——> 1
int a=10; ——> 5
Char b="abc"; ——> 9
int tc=30; ——> 6
Char d="xyz"; ——> 9
in /* Comment */t m=40.5 ——> 4
} ——> 1

(b) void main() ——> 4
{ ——> 1
int x=10, y=20; ——> 9
Char* a; ——> 4
a=&x; ——> 5
x=1xabj ——> 7
} ——> 1

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

"4 * + = c b a"

→ lexical analysis : identify tokens

"4 * + = c b a" → 7

Construct parse tree, Typechecking, Error handly.

→ Syntax analyzer

identifier: c, b, a, 4

operator: *, +, =

→ Intermediate code generator,

→ code optimizer

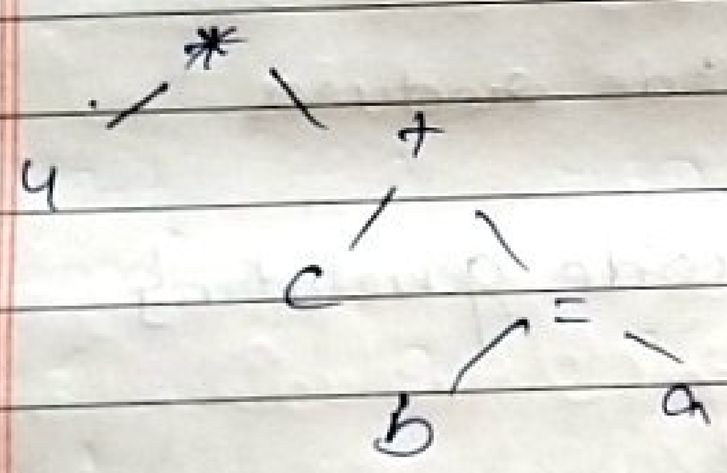
→ code generator.

→ Semantic analyser

a [+=] 4 * (c * b);

before this operator must available is present

4 * + = c b a



a + = (c * b) (semantic o/p)

t0 = (c * b)

t1 = * + 0

a + = t0t1

{ Intermediate Code generator }

a + = t0t1 → { code optimizer }

→ mov ax [c] { code generator }

movbx [b]

mul bx, ax

mov cx, 4

mul cx, bx

movdx, [a]

Add dx, cx

mov [a], dx

(4) write down the output of each phases for the Expression a := b + c * 50.

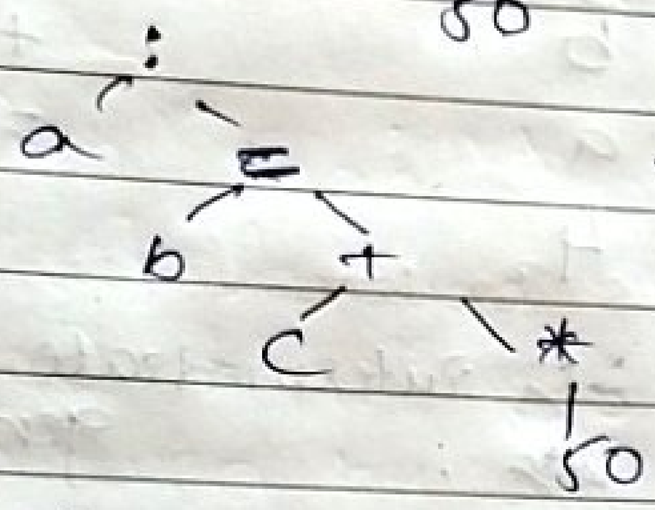
a := b + c * 50

→ lexical analysis

tokens: a, :=, b, +, c, *, 50.

→ Syntax analysis

Identifier	tokens
a	:=
b	+
c	*
50	



→ Semantic analysis
(type checking, error handling)

a = b + c * 50

→ Intermediate code generator

t0 = c * 50

t1 = b + t0

a = t0 + t1

→ code reducer

a = t0 + t1

→ LDA b, R1 {code generator}
LDA c, R2
Mul R2, 50, R2
ADD R1, R2, R3
STA R3, a
HLT

(5) Illustrate the output of each phase of compilation of the input "p = (a*b) + (b*c) ** 2".

p = (a*b) + (b*c) ** 2

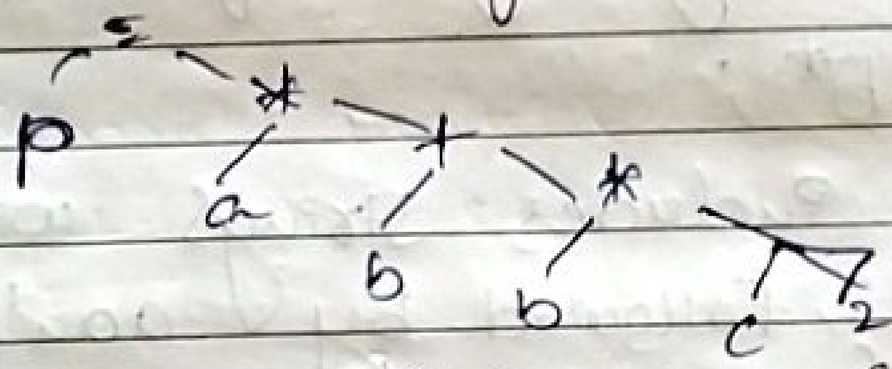
lexical analysis

(i) tokens => p = (a * b) + (b * c) * 2

(ii) Syntax analysis

Identifier	tokens
p	=
a	*
b	+
b	*
c	
2	

(iii) Semantic analysis (no errors)



(iv) Intermediate code generator

$t_1 = a * b$

$t_2 = b * c$

$t_3 = t_2 * c$

$p = t_1 + t_3$

(v) code reducer

$p = t_1 + t_3$ ✗

(vi) Code generator

LDA a, R1

LDA b, R2

MUL R1, R2, R3

LDA b, R4

LDA c, R5

MUL R4, R5, R6

MUL R6, R1, R7

ADD R3, R7, R8

STA R8, P

HLT

(6) Describe the languages denoted by the following regular expressions:

(a) $(a+ab)^*$ (b) (1^*0^*) (c) $(b^*(aaa)^*b^*)$

(d) $100(1+0)^*101$ (e) a^*b^*c

(a) $(a+ab)^*$

Strings accepting any combination of a & ab & it allows repetitions of a (or) ab including ε

(b) 1^*0^*;

Strings accepting any no. of 1's & 0's.

(c) $b^*(aaa)^*b^*$;

This language includes strings with zero's (or) more b's followed by aaa's & again followed by zero's (or) more b's

Eg: baaabbb bbbaaab

(d) $100(1+0)^*101$

Starts with 100 followed by combination of any no. of 1's (or) 0's but ending with 101.

(e) a^*b^*c

language consists of strings with any combination of a's & b's & c's including empty set (ε).

(7) Illustrate the compiler internal representation of the changes in the source program, as translation progresses by considering the translation of the statement $(x=a*b-c)$ [x, a, b, c - real].

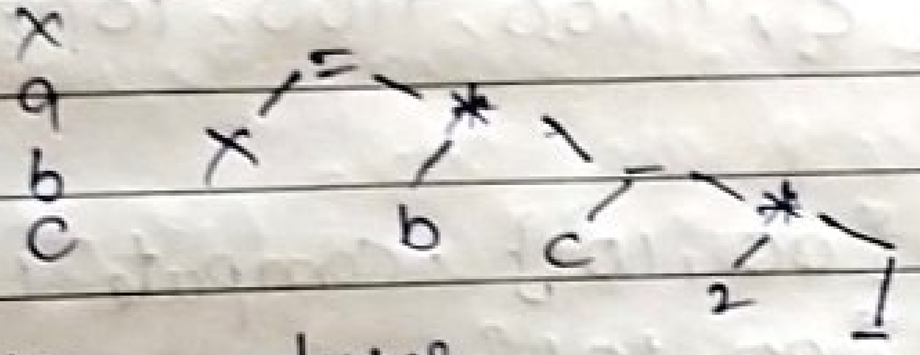
$(x=a*b-c*2)$ [x, a, b, c - real]

(i) lexical analysis (tokenization)

$x = a * b - c * 2$ x a b c - real

(ii) Syntax analyzer

Identifier	Operator
x	=
a	*
b	-
c	*
2	-
x	
a	
b	
c	



(iii) Semantic analyzer

$x = (a * b - c * 2)$ [x, a, b, c - real]

no errors / no type checking.

(iv) Intermediate code generator

$t_1 = a * b$

$t_2 = c * 2$

$x = t_1 - t_2$

(v) Code reduced

$x = t_1 - t_2$ ✗

(vi) Code generator

```
LDA  a, R1
LDA  b, R2
MUL  R1, R2, R3
LDA  c, R4
MUL  R4, 2, R5
SUB  R3, R5, R6
STA  R6, X
HLT
```

(8) Let $L_1 = \{11, 00\}$, $L_2 = \{1100, 101\}$, Compute $L_1 L_2$ & L_1^*.

$L_1 = \{11, 00\}$, $L_2 = \{1100, 101\}$

Compute $L_1 L_2$ & L_1^*

↓ Con Catenation ↓ Kleen closure

$L_1 L_2 = \{1100, 101, 0011, 0000\}$

$L_1^* = \{\varepsilon, 11, 00, 1100, 101, 1111 \cdots\}$

(9) let $L_1 = \{00, 101\}$ · compute L_1^*

$L_1 = \{00, 101\}$.

$\Rightarrow \{00, 101, 100, 010 \cdots\}$

(10) let $L_1 = \{aab, bb\}$, $L_2 = \{babb, aa\}$.

Compute $L_1 L_2$ & L_1^*

↓ Contacten ↓ Kleen Closure

$L_1 L_2 = \{aabbabb, aababa, bbbab, aabb\}$

$L_1^* = \{aab, bb, aabbb, aa bab \cdots\}$.

(11) ~~let $L_1 = \{aab, bb\}$ $L_2 = \{babb, aa\}$ · Compute $L_1 L_2$ & L_1^*~~

(12) write regular expressions for the following languages of all strings in $\{0,1\}^*$.

(i) Strings that do not end with 01.

(ii) Strings with odd number of 1's.

$\Sigma = \{0, 1\}$

(i) Strings don't end with 01

$(0^* + 1^* + (00 + 11)^*)^*$

any combination of 0's & 1's (or) pairs of 00's (or) 11's but not end with 01

(iii) strings with an odd numbers of 1's.

$\Sigma = \{0,1\}$

$0^*(1(01)^*0+1)^*$

any combination of 0's followed by an odd number of 1's including the case where 1's are seperated by 01 repeated zero (or) more times.

(13) write regular expressions for the following languages of all strings on $\{0,1\}^*$.

(i) strings that start with 1 & do not end with 10.

(ii) strings with length 6 (or) less.

(i) strings that starts with 1 & don't end with 10 $\Sigma = (0,1)$

$1(0^*+1^*(00+11)^*0^*+1$

(ii) strings with length 6 (or) less.

$\Sigma = \{0,1\}$ $(0+1)\ 0,6$

strings consisting of 0's & 1's with a length of 6 (or) less.

(14) Find a regular expression corresponding to each of the following subsets of $\{0,1\}^*$.

(i) the language of all strings that dont contain the substring 110.

(ii) the language of all strings that contain both 101 & 010 as substring.

(iii) the language of all strings in which both the number of 0's & number of 1's are odd.

(q) language of all strings dult contain the substring 110.

$$(0^*1^*)^* + (0+1)^*$$

(qq) language of all strings that contain both 101 & 010 as substring

$$(0+1)^* 101 (0+1)^* 010 (0+1)^*$$

(qqq) language of all strings in which both the no. of 0's & 1's are odd.

$$(0^*10^*1)^*0^* + (1^*0^*0)^*1^*$$

12 (i) strings containing an odd no. of 0's

$$\Sigma = \{0,1\}$$

$$(1^*01^*0)^*1^*$$

(ii) begin or end with 00 (or) 11

$$(00+11)(01)^* + (1+0)^*00 + (1+0)^*11$$

15 (i) language of all strings containing at least two 0's $\Sigma = \{0,1\}$.

$$(1^*0)2, 1^*$$

(ii) language of all strings containing both 101 & 010 as substrings.

$$(0+1)^*100(0+1)^*010(0+1)^*$$

(iii) language of all strings not ending with 01.

$$(0+1)^*(00+0+11+1).$$

16 (i) Begins with 00 & Ends with 11

$$00(0+1)^*11$$

(ii) Contains alternate 0 & 1.

$$(01)^*$$

17 (i) Begin or End with 00 & 11

$$00(0+1)^* + (0+1)^*00 + 11(0+1)^* + (0+1)^*11$$

(iii) language of all strings contains 11 & 010 as substrings.

$(0+1)^* 11 (0+1)^* 010 (0+1)^*$

(18) (i) language of all strings that begin (or) end with 11 (or) 00

$11(0+1)^* + (0+1)^* 11 + 00(0+1)^* + (0+1)^* 00$

(ii) begin with 1 & end with 0.

$1(0+1)^* 0$

(19) (i) $(0+1)^*$

strings accepting the combination of any no. of 0's (or) 1's.

(ii) $(01)^*$

String accepting the combination of any no. of 0's & 1's.

(iii) $(0+1)$

String accepting 0 (or) 1

(iv) $(0+1)+$

String accepting 0 (or) 1 without an emptyset.

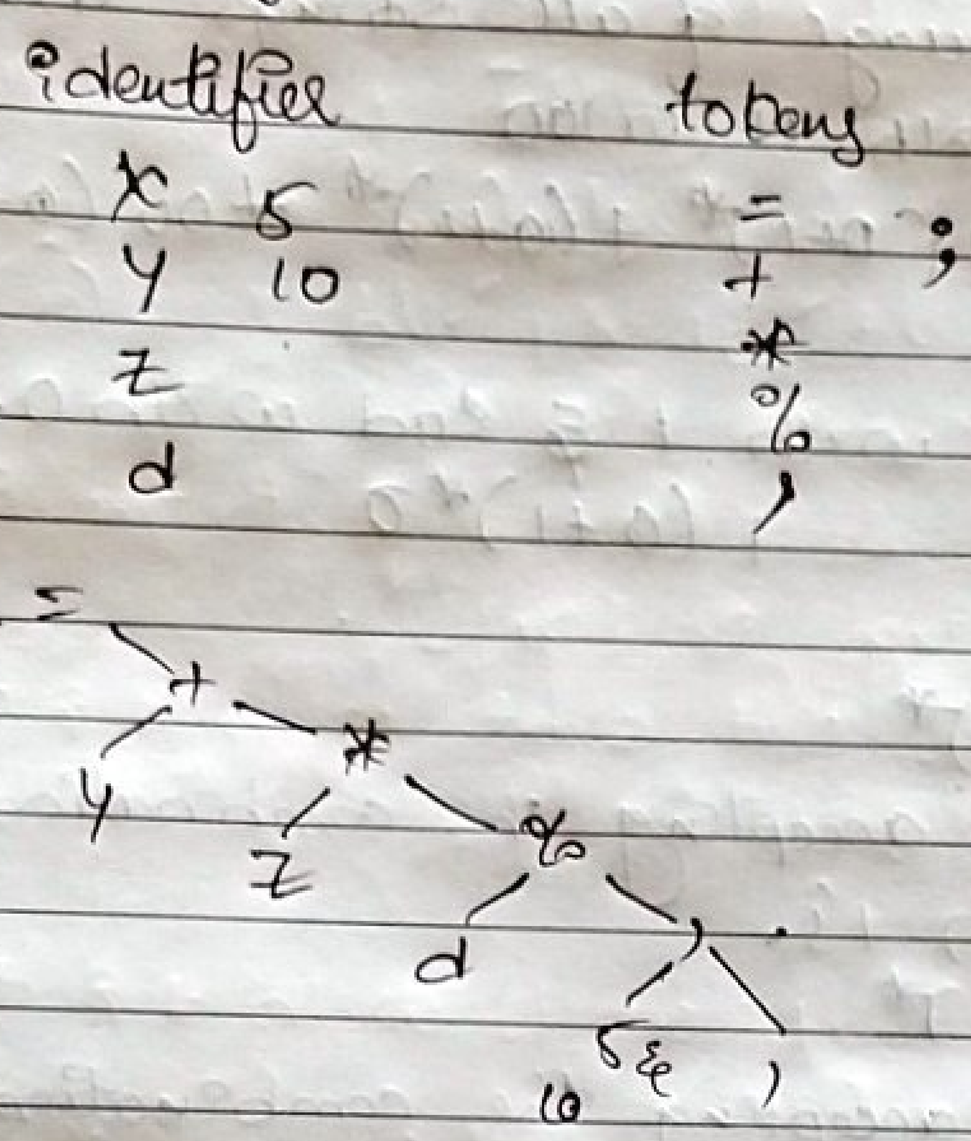
(20)
```
Int x=5, y=10, z;
z = x+y*5;
printf("%d", z);
```

(i) lexical analyser

identifying tokens

Int , x, =, 5, y, =, 10, z, ;, z, =, x, +, y,
*, 5, ;, printf, (, ", %d, ", z,);.

(ii) Syntax analyser.



(iii) Semantic analyser
(no errors, no typechecking)

(iv) Intermediate code generator.

$t_1 = 5$
$t_2 = 10$
$x = t_1$
$y = t_2$
$t_3 = y * 5$
$t_4 = x + t_3$
$z = t_4$

(v) Code reducer

$z = t_4$ X

(vi) Code generator

```
x addr
y add 10
z add 0
Mov ax, 5
Mov [x] ax
Mov ax, 10
Mov [y], ax
Mov ax, [y]
```

```
MUL ax, 5
add ax, [x]
Mov [z], ax
Mov ax, [z]
format db
"%d", 0
```