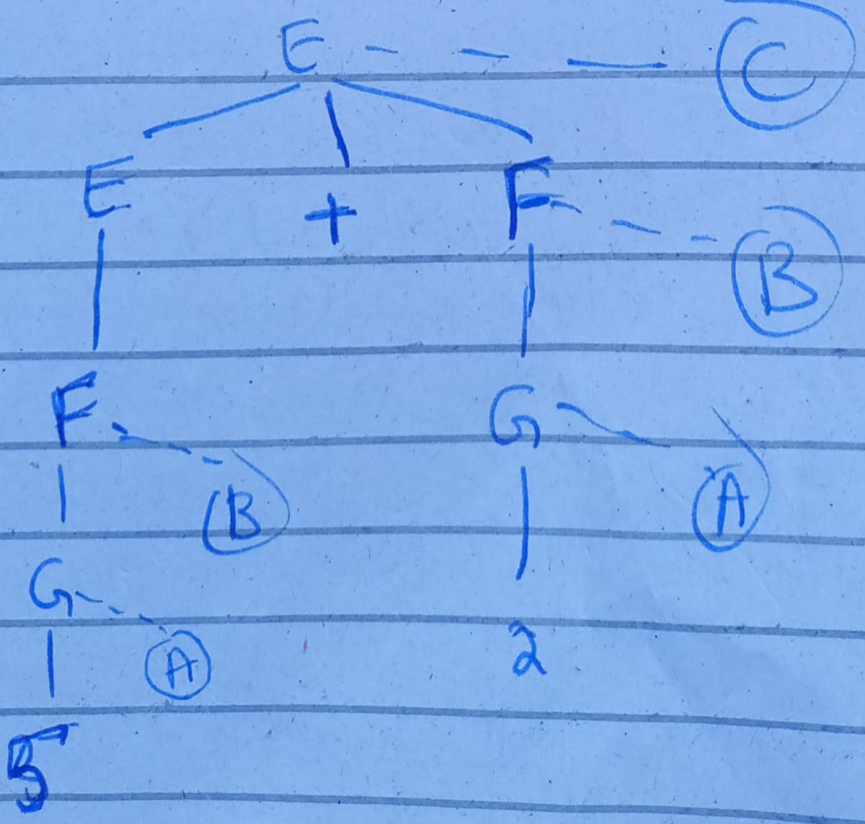
$E \rightarrow E + F \mid F$

$F \rightarrow F * G \mid G$

$G \rightarrow lit$

⇒ Convert to Postfix using SDT

5 * 2 * 3



Infix to Postfix
" " Prefix
Postfix to Infx
" " Prefix

(A) G.val = lit = 5

(B) F.val = G.val
= 5

(C) $E_p$.val = + $E_c$.val F.val

= + 5 2

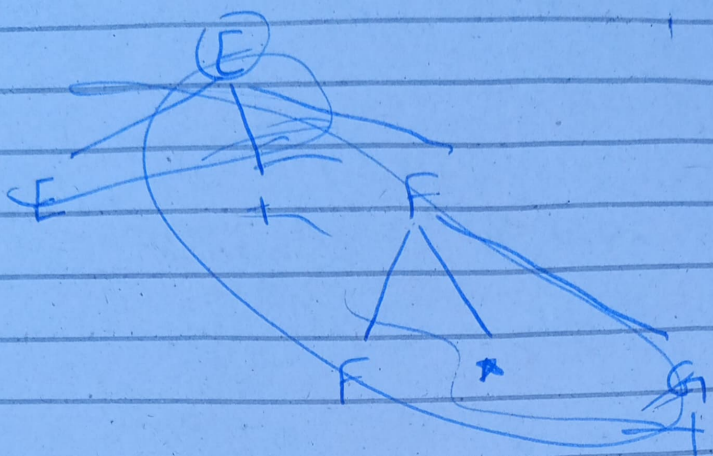
2 x 5 + 8.

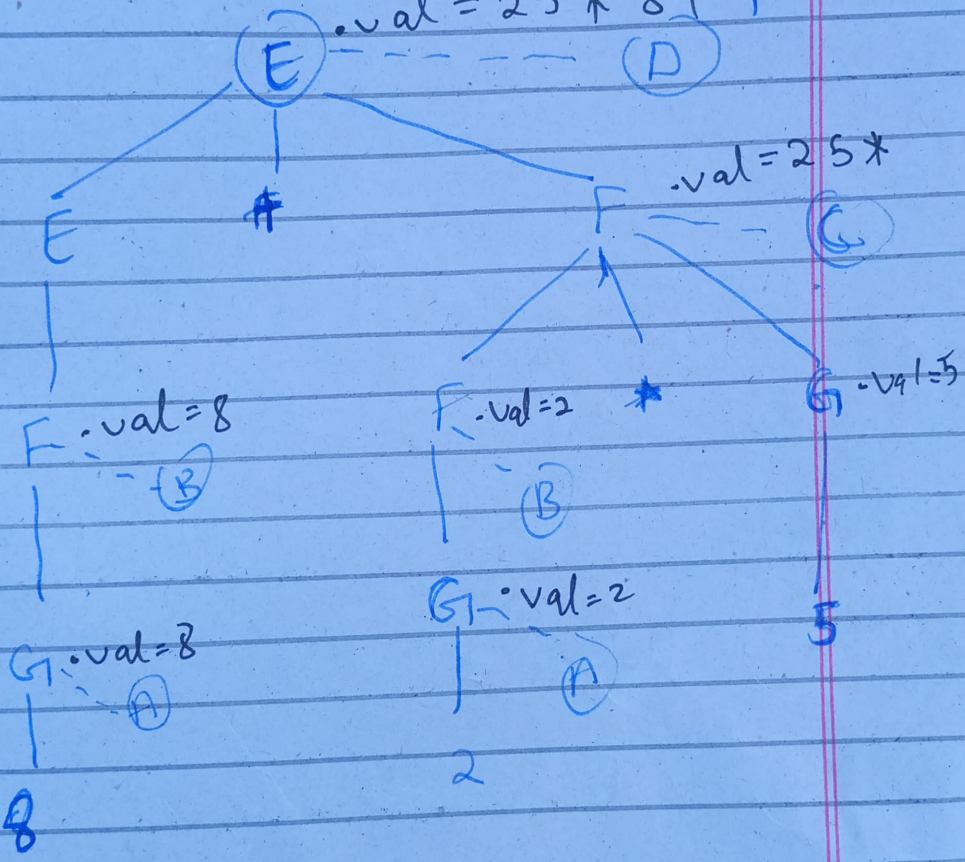
+ * 2 5 8

Prefix.

Post

2 5 * 8 +



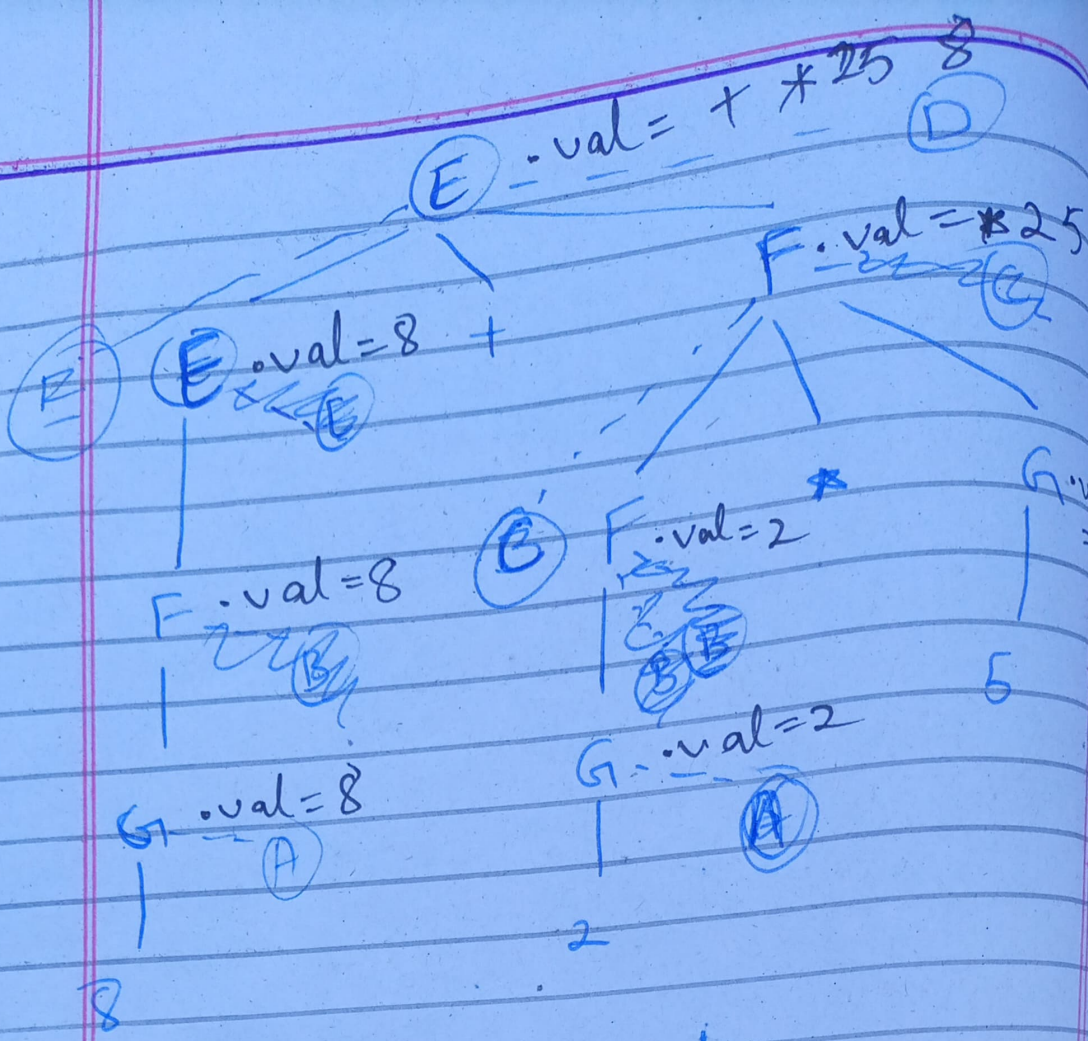




(A) G.val = lit = 8

(B) F.val = G.val = 8

(C) $F_p$.val = $F_c$.val G.val *

(D) $E_p$.val = F.val $E_c$.val +




(A) G.val = lit

(B) F.val = G.val

(C) $F_p$.val = * $F_c$.val G.val

(D) $E_p$.val = + F.val $E_c$.val

(E) E.val = F.val

(A) Print(lit.val)

(B)

(C) print(*)

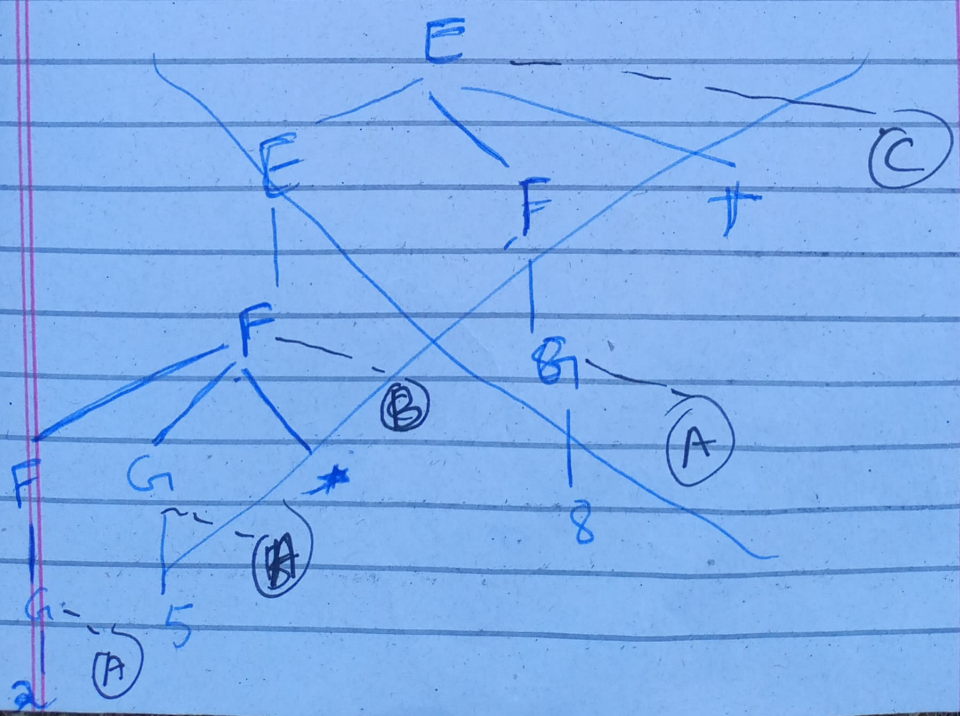
(E) Print(+)

⟷

Postfix Gramar

E → E F + | F
F → F G * | G
G → lit

2×5 + 8 → 2 5 * 8 +

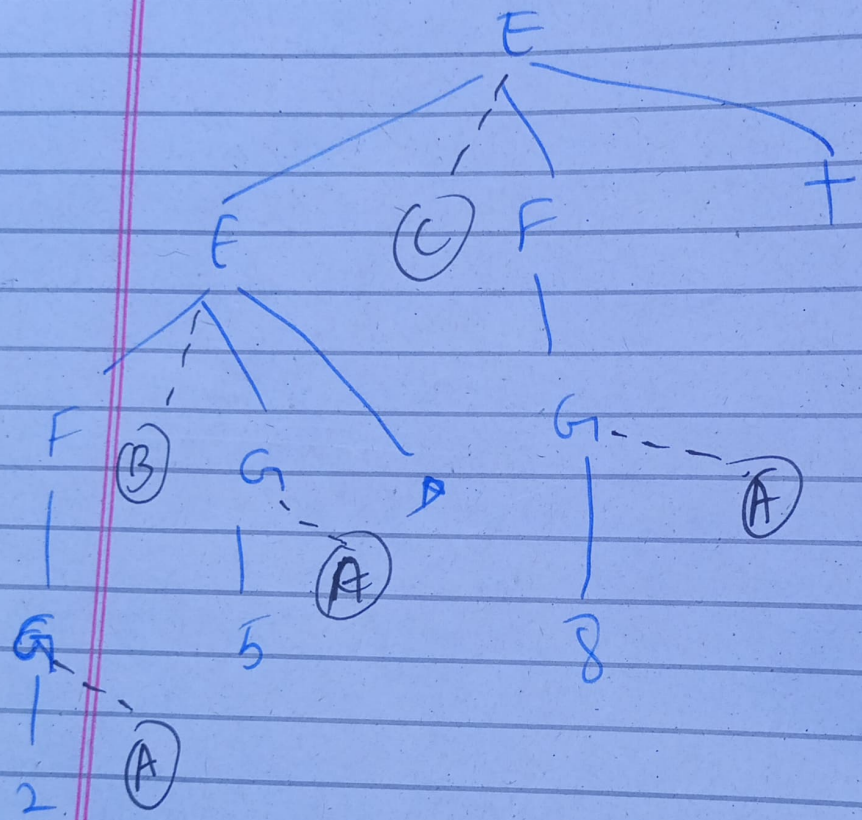


(A) Print (G.lit)

(B) Print (*)

(C) Print (+)

2 × 5 + 8

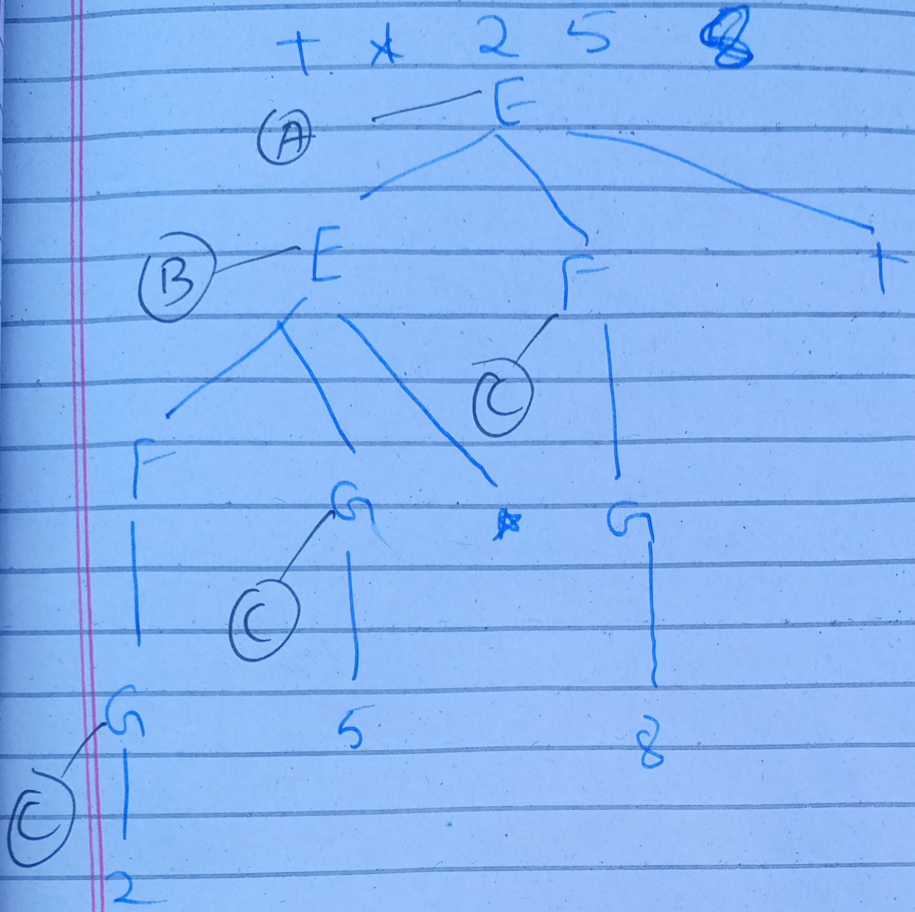



(A) print (G.lit)

(B) print (*)

(C) print (+)

# Postfix to Prefix




A print (+)

B print (*)

C print (G.lit)

# Right Recursive Grammar

$E \rightarrow TE'$

$E' \rightarrow +TE' \mid \epsilon$

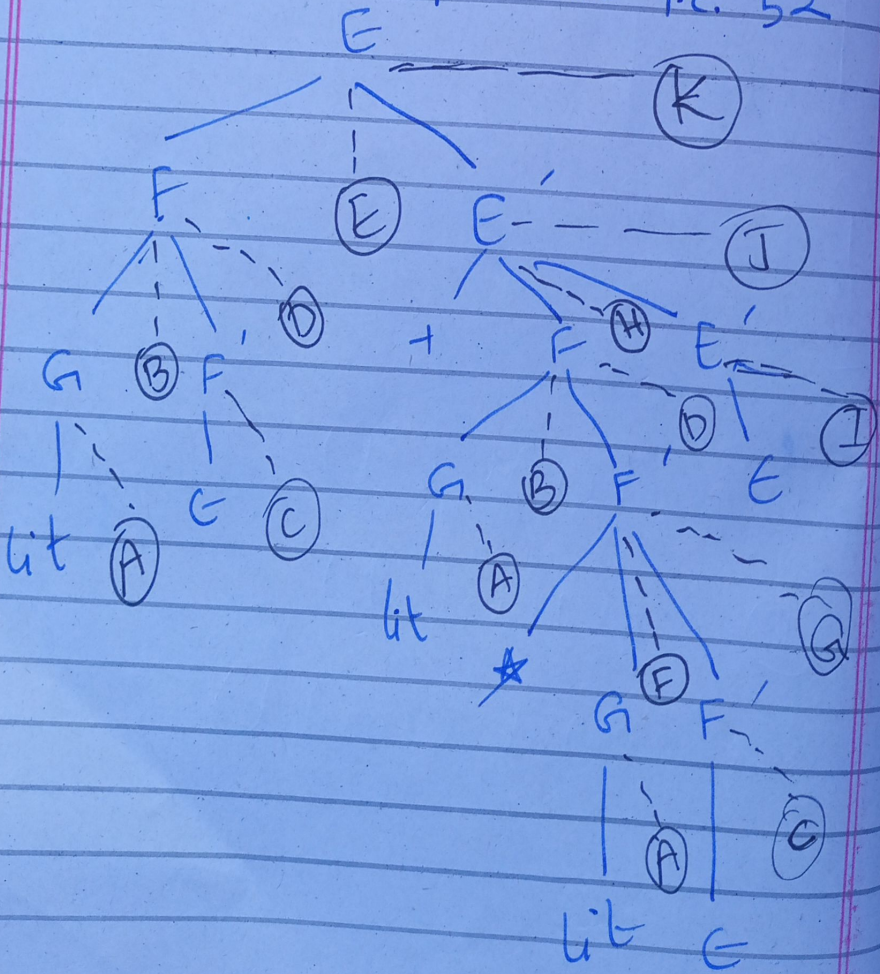
$T \rightarrow GF'$

$F' \rightarrow *GF' \mid \epsilon$

$G \rightarrow lit$

$2+5\times10$

Evaluate expression i.e. 52



In previous examples we passed from child to parent
⟹ Synthesized variables

Now we have to pass from parent to child
⟹ Inherited variable.

Note

* Passing from child to parent (Both variables must be synthesized)

* Passing from parent to child ( one can be inherited and other synthesized)

(A) G.svar = lit.sval

(B) F'.ival = G.sval

(C) F'.sval = F'.ival

(D) F.sval = F'.sval

(E) E'.ival = F.sval

(F) $F'_c.ival = F'_p.ival * G.sval$

(G) $F'_p.sval = F'_c.sval$

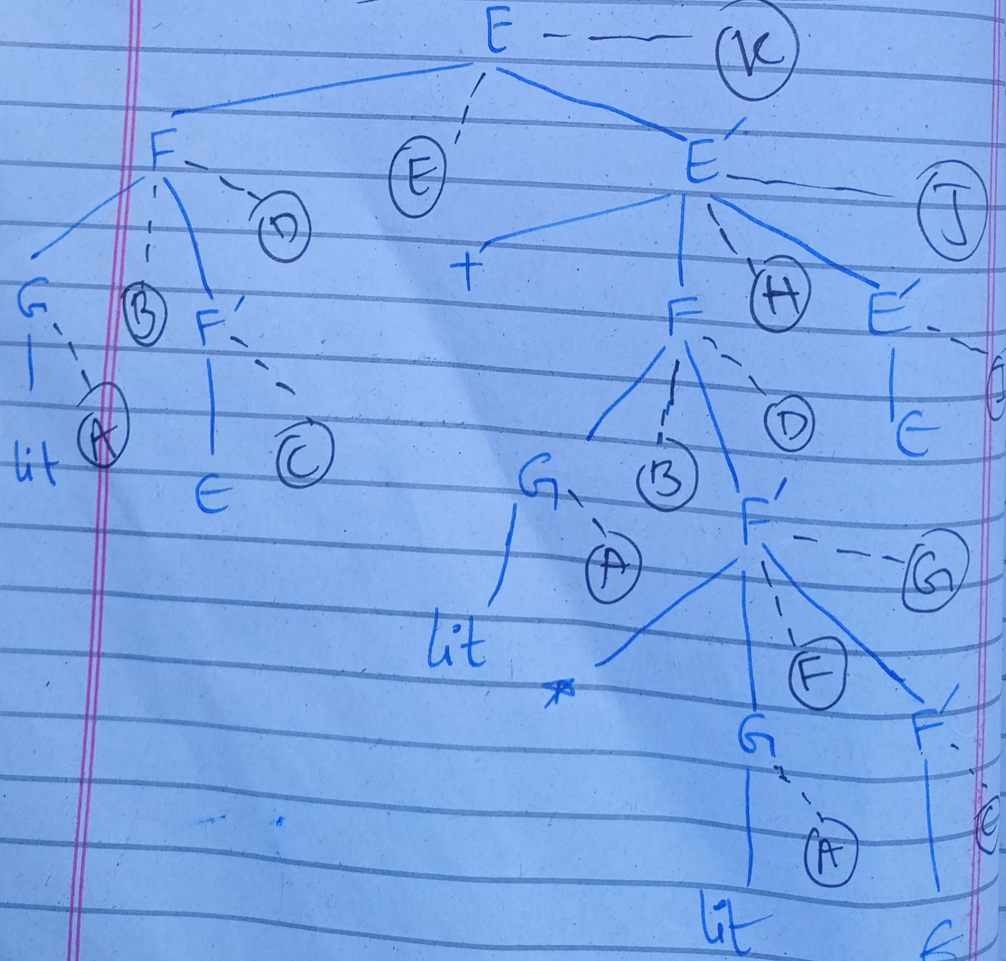
(H) $E'_c.ival = E'_p.ival + F.sval$

(I) $E'.sval = E'.ival$

(J) $E'_p.sval = E'_c.sval$

(K) $E.sval = E'.sval$

## Infixt to Postfix



2 + 5 * 10 → 2 5 10 * +

(A) G.sval = lit.sval

(B) F'.ival = G.sval

(C) F'.sval = F'.ival

(D) F.sval = F'.sval

(E) E'.ival = F.sval

(F) $F'_c$.ival = $F'_p$.ival G.sval *

(G) $F'_p$.sval = $F'_c$.sval

(H) $E'_c$.ival = $E'_p$.ival F.sval +

(I) E'.sval = E'.ival

(J) $E'_p$.sval = $E'_c$.sval

(K) = $E'_p$.sval = $E'_c$.sval

Note

use istr & $str instead of val.

Dec → Bin
(IMP)

# Binary to Decimal

$B \rightarrow BZ \mid \epsilon$

$Z \rightarrow 0 \mid 1$

$B \rightarrow Z \mid 0 \mid 1$

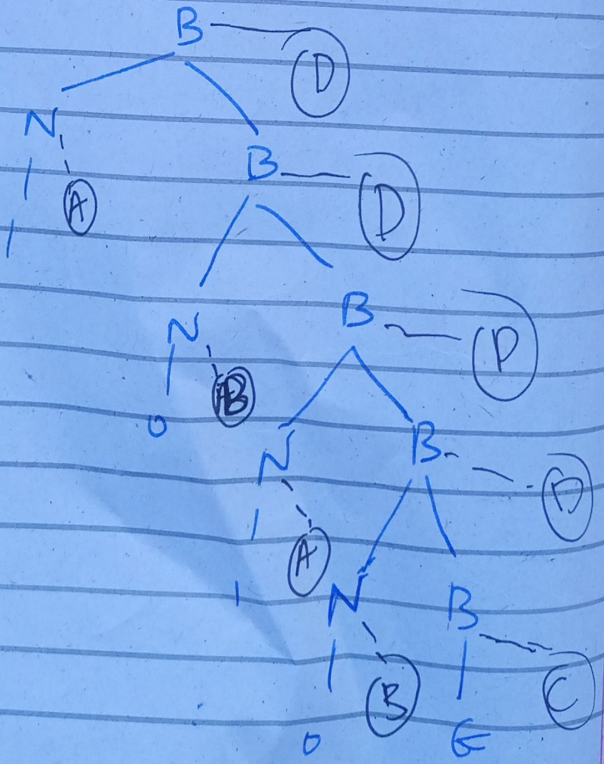
$Z \rightarrow Z0 \mid Z1 \mid \epsilon$

$B \rightarrow 0B \mid 1B \mid 0 \mid 1$

$B \rightarrow NB \mid \epsilon$

$N \rightarrow 0 \mid 1$

1010



(A) $N.sval = 1$

(B) $N.sval = 0$

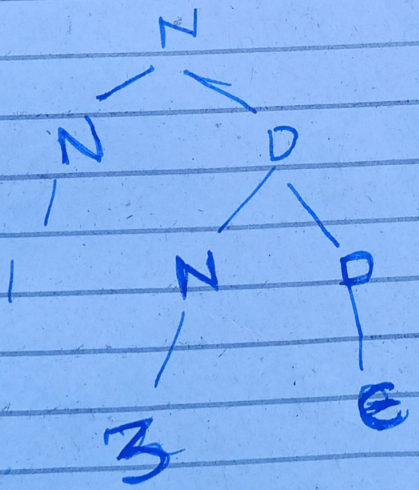
(C) $B.sval = 0$
$B.spos = 0$

(D) ~~$B.sval = N.sval + B.sval$~~
$B.spos = B_c.spos + 1$
$B.sval = N.sval \times 2^{B.spos} + B_c.sval$

Decimal → Binary

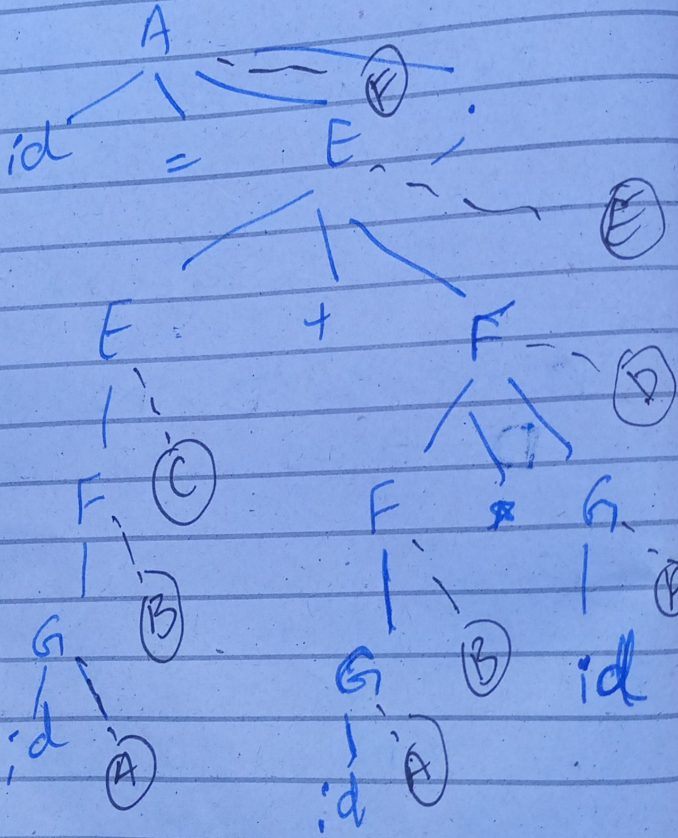
$N \rightarrow D\ N \mid \epsilon$
$D \rightarrow 0 \mid 1 \mid 2 \ldots$

13



id = id + id * id -

A → id = E ;
E → E + F | F
F → F * G | G
G → id




(A) G.type = getTypeST(id);

(B) F.type = G.type

(C) E.type = F.type

(D) F.type = Greater(F_c.type, G.type)

(E) E_p.type = Greater(E_c.type, F.type)

(F) if (id.getTypeST(id) ≠ E.type)

{

Error();

}

## Right Recursive

A → id = E;

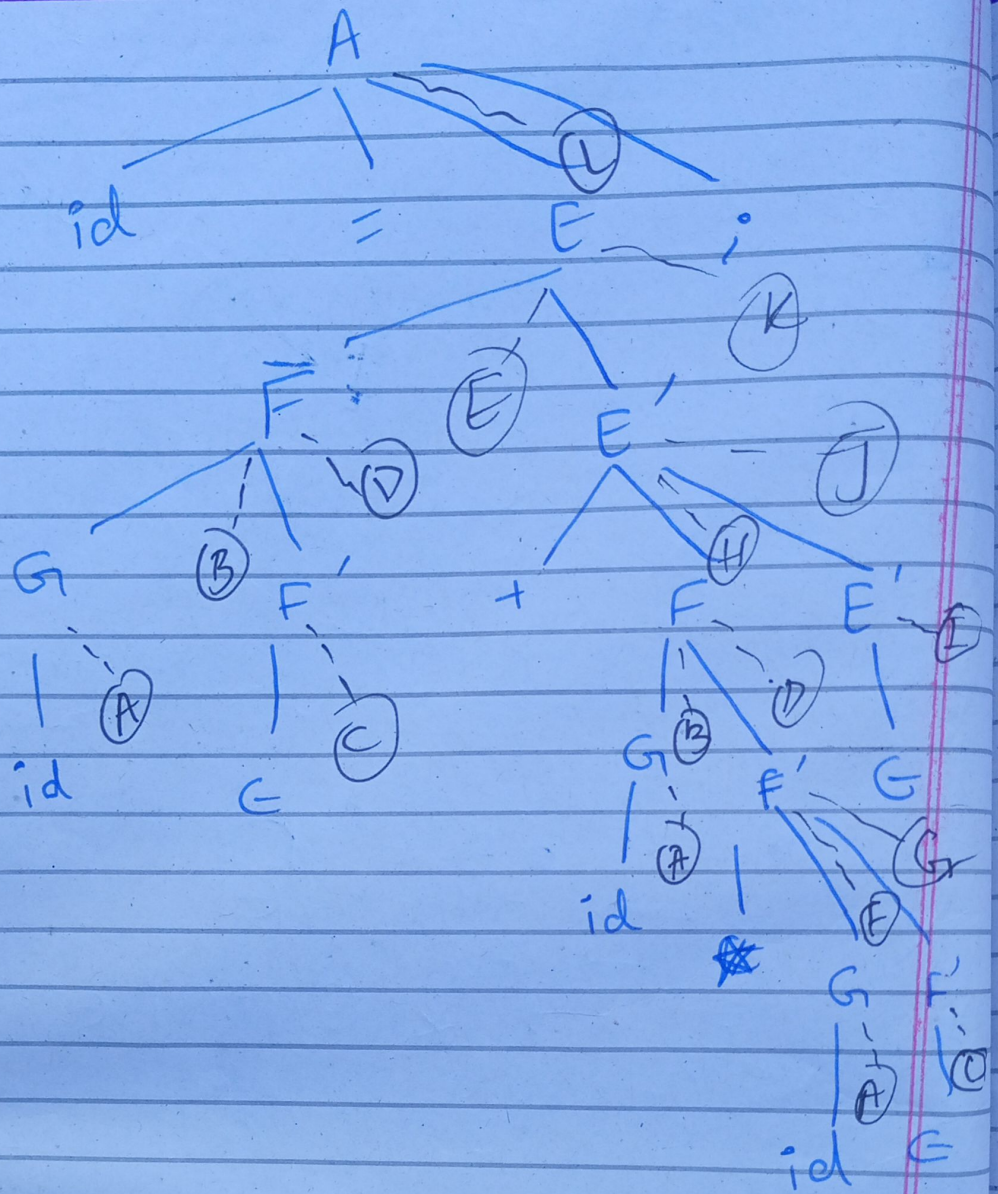
E → F E'

E' → +F E' | ∈

F → G F'

F' → * G F' | ∈

G → lit




(A) G.stype = getTypeST(id);

~~(B) F'.stype = G.stype~~

(B) F'.itype = G.stype

(C) F'.stype = F'.itype

(D) $F_p$.stype = $F_c$.stype

(E) $E'.itype = F.stype$

(F) $F'c.itype = Greater(F'_p.istype, G.stype)$

(G) $F'_p.stype = F'_c.stype$

(H) ~~$E'_c.istype = E'_p.is$~~

(A) $E'_c.itype = Greater(E'_p.itype, F.stype)$

(I) $E'.stype = E'.itype$

(J) $E'_p.stype = E'_c.stype$

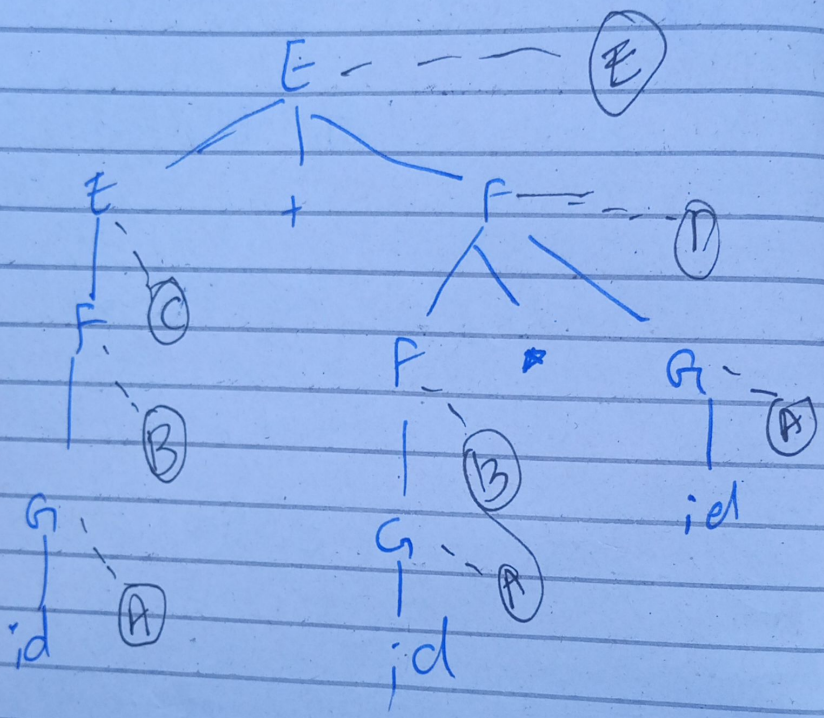
(K) $E.stype = E'.stype$

(L) ~~$A.stype = Greater$~~

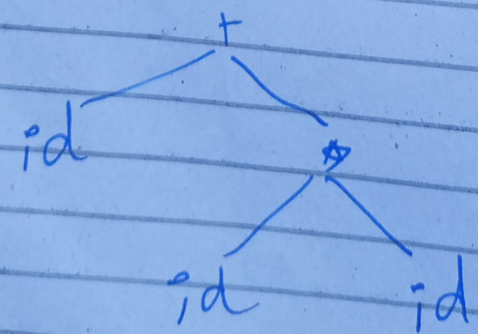
```
if( getTypesT(id) (E.type)
    {
        Error();
    }
```

id + id * id

E → E + F | F
F → F * G | G
G → id




Abstract syntax tree



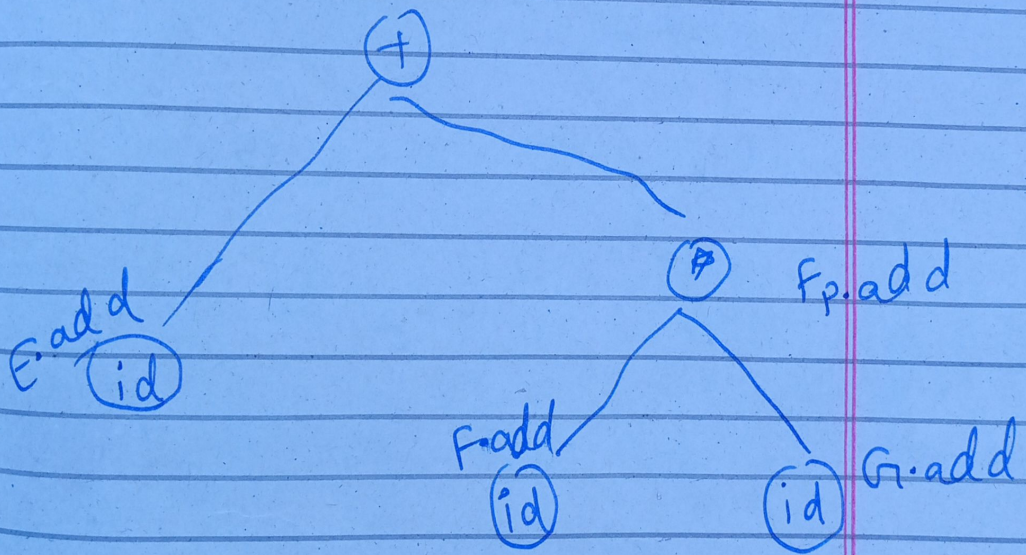
(A) G.add = new Leaf(id)

(B) F.add = G.add

(C) E.add = F.add

(D) $F_p$.add = new Node('*', $F_c$.add, G.add)

(E) $E_p$.add = new Node('+', $E_c$.add, F.add)



# Right Recursive

$E \rightarrow FE'$

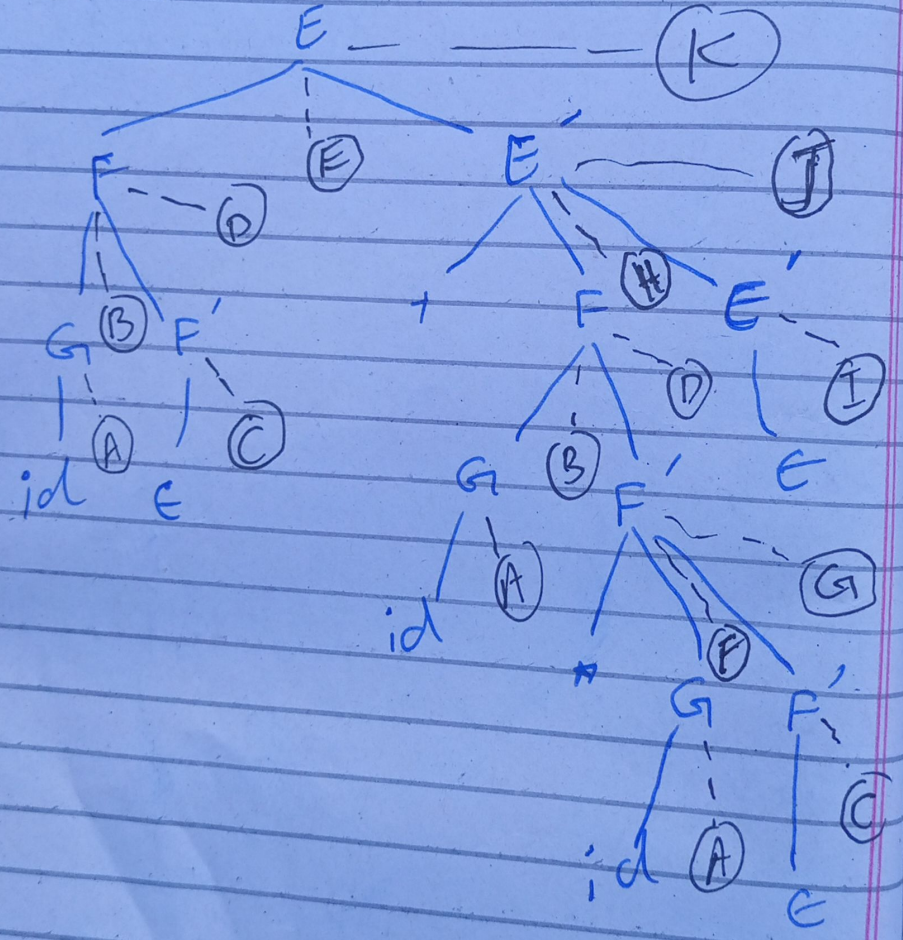
$E' \rightarrow +FE' \mid \epsilon$

$F \rightarrow GF'$

$F' \rightarrow *GF' \mid \epsilon$

$G \rightarrow id$

id + id * id



new Leaf(id)

(B) $F.iaddr = G.saddr$

(C) $F'.saddr = F'.iaddr$

(D) $F_p.saddr = F'_p.saddr$

(E) $E'.iaddr = F.saddr$

(F) $F'_c.iadr = new\ Node('*', F_{p}.iaddr, G.saddr)$

(G) $F'_p.saddr = F'_c.saddr$

(H) $E'_c.iaddr = new\ Node('+', E_p.iddr, F.saddr)$

(I) $E'.saddr = E'.iaddr$

(J) $E_p.saddr = E'.stype$

(K) $E.saddr = E'.saddr$

# 3 Address Code

$x = y + z + a$ ,

$t_1 = y + z$ ⟶ User generated Address

$t_2 = t_1 + a$

$x = t_2$

↳ Compiler generated address.

```
if( x==y || y<3)
  {


  }
```

In three address code we use conditional & unconditional jumps

```
if x==y  Jump L1
Jump L2
L2 : If y<3 Jump L1
     Jump L3
L1 :    |
        code
        :
L3:     :
```

```
if (x == y && y < 3) {

}
```

```
if x == y JMP L2
   JMP L3
L2: if y<3 JMP L1
   JMP L3

L1:  ;
     ;
     code

L3:  ;
     ;
     ;
```

```
for (int i=1, j=0; i<20 && j>30;
                   i++, j++)
{

}
```

```
INT  i=1
INT  j=0

L1: IF  i < 20  JMP  L2
    JMP  L4

L2  IF  j > 30  JMP  L3
    JMP  L4

L3
    :
    Code
    :
    ~~JMP  L1~~
    i = i-1
    j = j+1
    JMP  L1

L4
    |
```

structure for storing 3-Add code

Quad ruple

| OP | res | OP1 | OP2 |
|---|---|---|---|
| = | i | 1 | |
| = | j | 0 | |
| < | L2 | i | |
| JMP | L4 | | 20 |
| > | L3 | j | |
| JMP | L4 | | 30 |
| - | i | i | 1 |

After this we run optimazition to remove unnecessary code like

$$-t_1 = y + z$$
$$-t_2 = -t_1 + 9$$
$$x = -t_2$$

we can optimize it like

$$-t_1 = y + z$$
$$x = -t_1 + 9$$

# Note for Arrays

Int arr [3] = {2,4,5}

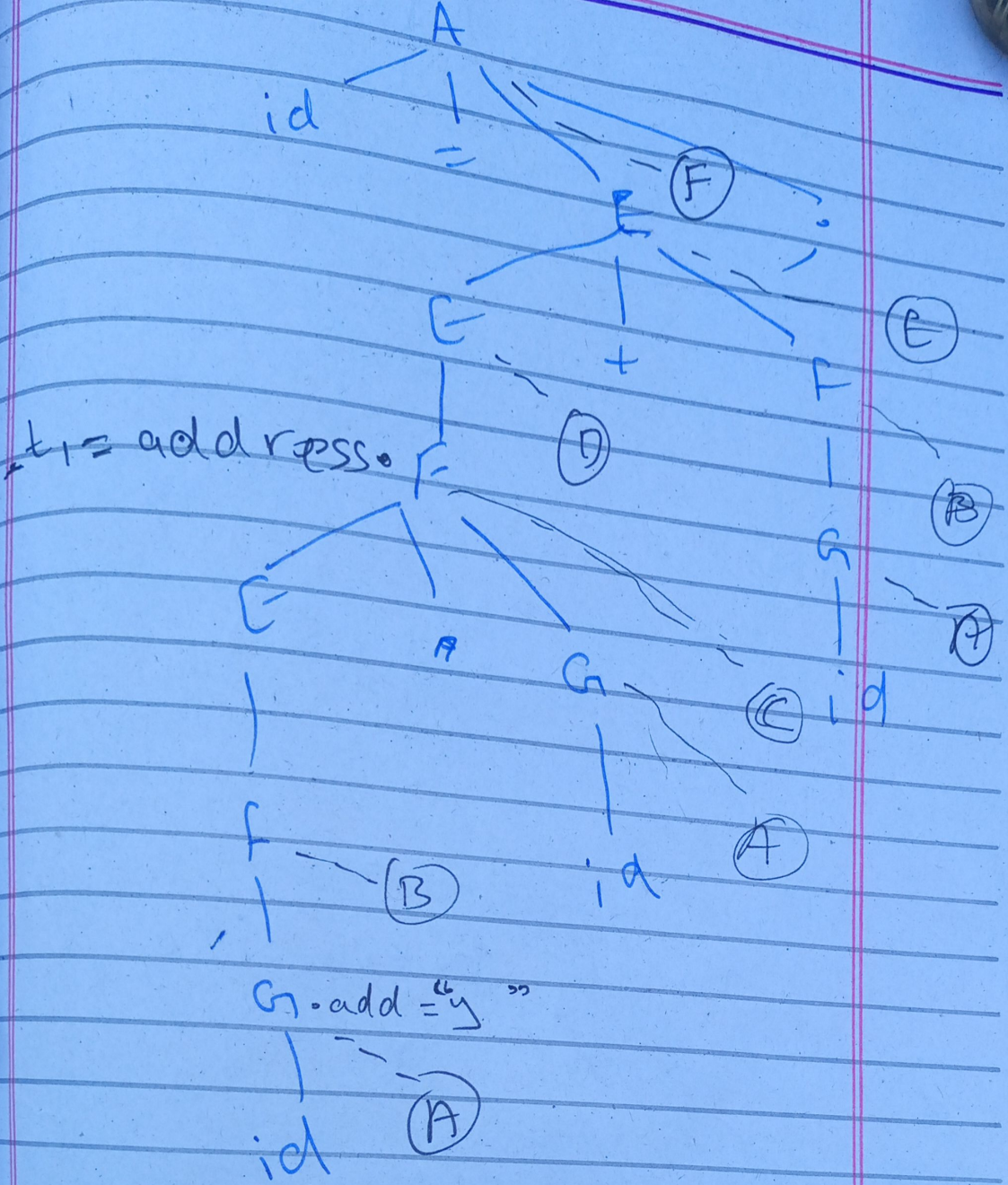
* ptr = arr [3]
* *ptr = 2
* ptr++
* *ptr = 4
* ptr++
* *ptr = 5

x = y * 2 + q
id = id * id + id

A → id = E ;
E → E + F | F
F → F * G | G
G → id

Three addr code

$t_1 = y * 2$
$t_2 = t_1 + q$
$x = t_2$




t1 = address.

(A) G.address = getLexim(id.key)

(B) F.address = G.address

(C) Fp.address = GetTemp(Address());
gen3AddressCode("+", Fp.address, Fc.address, G.address)

(D) E.address = F.address

(E) Ep.address = getTempAddress

gen3AddressCode('+', Ep.address,
Ec.address,
F.address);

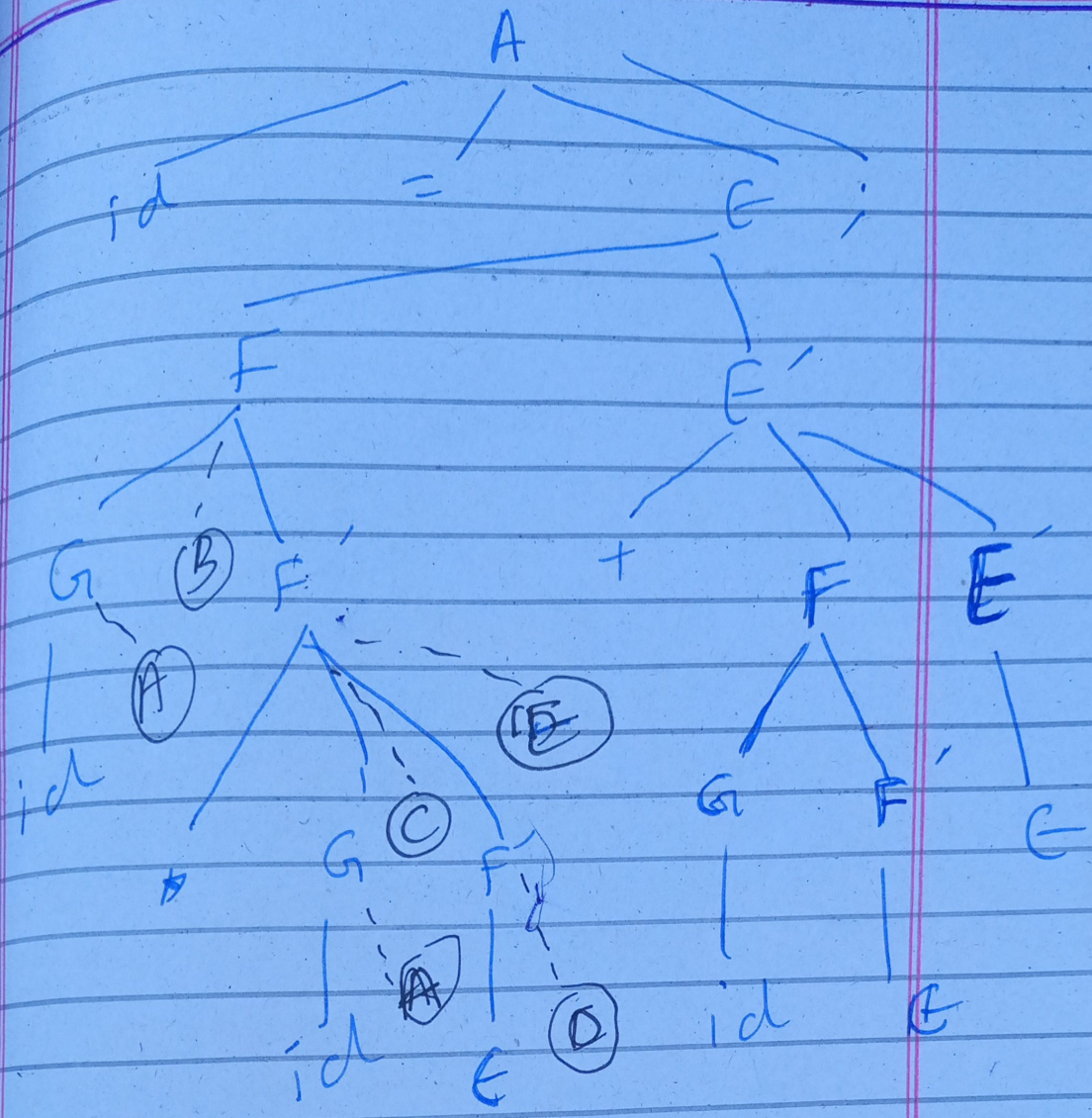
(F) gen3AddressCode('=',
getLexim(id.key),
E.address);

⟷

Right Recursive

A → id = E;
E → FE'
E' → +FE' | ∈
F → GF'
F' → *GF' | ∈
G → id

id = id * id + id;




(A) $G.saddr = getLexim(id.key)$

(B) $F'.iaddr = G.saddr$

(C) $F'_p.saddr = getTemplexim()$
$gen3Addr(code('*'), F'_p.saddr, F'_p.iaddr, G.saddr)$

(E) ~~$F'_p.saddr = getTemplexim()$~~
~~$gen3AddrCode('*', F'_p.saddr, F'_p.iaddr, F'_c.saddr)$~~