Module IV

→ Syntax Directed Translation is a generalisation of a context free grammar in which each grammar symbol has an associated set of attributes, and partition into 2 subsets called synthesized and inherited attributes of that grammer symbol.

Grammer + Semantic Rules.

Grammar :

E ⇒ E+T / T

T ⇒ T*F / F

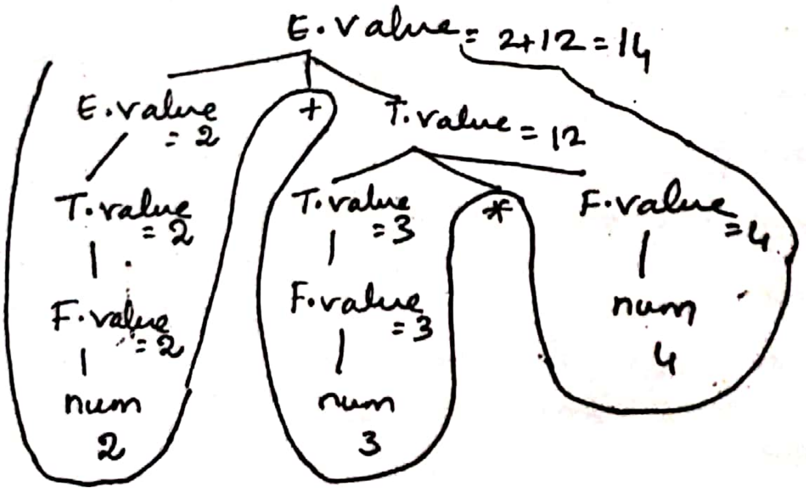
F ⇒ num

Raunteas

E ⇒ E+T	{E.Value = E.value + T.value}
E ⇒ T	{E.Value = T.value}
T ⇒ T*F	{T.Value = T.value * F.value}
T ⇒ F	{T.Value = F.Value}
F ⇒ num	{F.Value = num.lvalue}

lexical value.

Consider the expression, 2+3*4



Infix to postfix

E → E + T { printf("+"); } ①

E → T { } ②

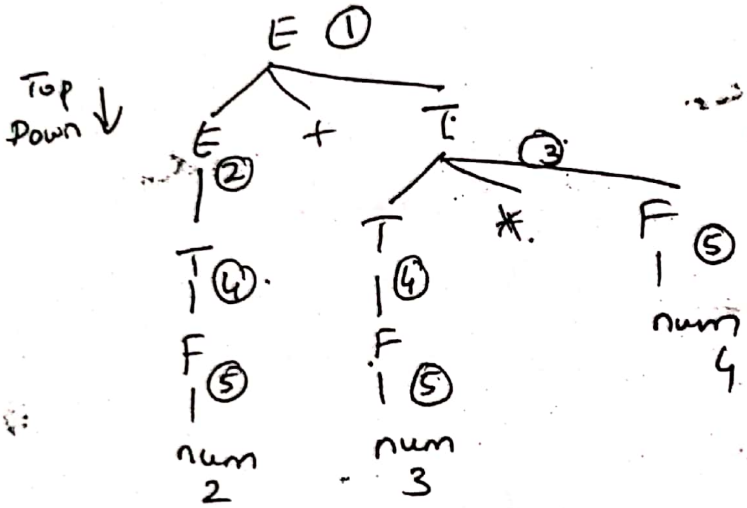
T → T * F { Printf("*"); } ③

T → F { } ④

F → num { printf(num.lval); } ⑤

Consider the expression 2+3*4 (infix). Convert into postfix

Top Down parsing.



Postfix expression : 2 3 4 * +

To build abstract syntax tree

E → E+T {E.nptr = mknode(E.nptr, '+', T.nptr)}

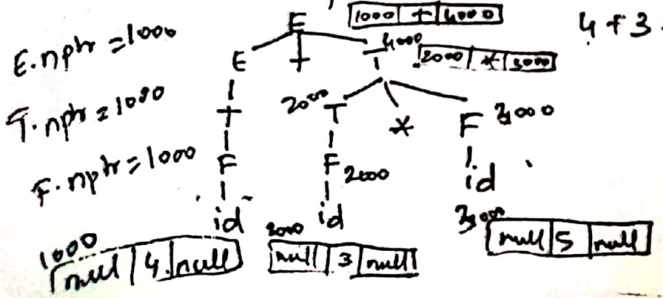
E → T {E.nptr = T.nptr;}

T → T*F {T.nptr = mknode(T.nptr, '*', F.nptr)}

T → F {T.nptr = F.nptr;}

F → id {F.nptr = mknode(null, idname, null)}

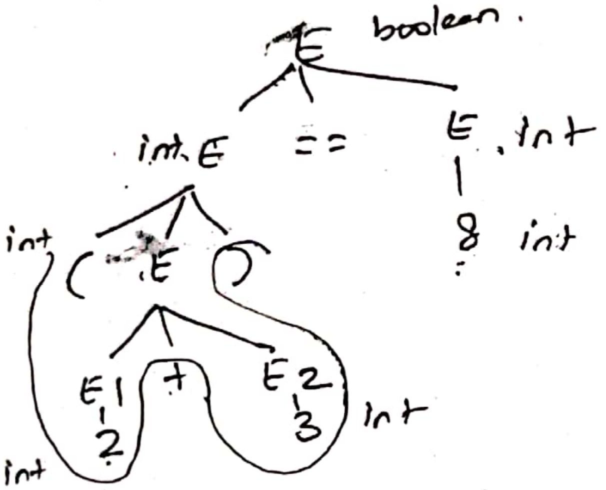
4+3*5



r for type check

```
E → E1 + E2 |   {if ((E1type == E2 type) && (E1 type = int)) then
                      Etype = int else error}
  E1 == E2 |    {if ((E1 type == E2 type)) && (E1 type = int / boole
  (E1) |                 then Etype = boolean else error}
                {E.type = E1.type;}
  num |         {E.type = int ;}
  True |        {E.type = boolean ;}
  False         {E.type = boolean ;}
```

Expression: (2+3) == 8 .

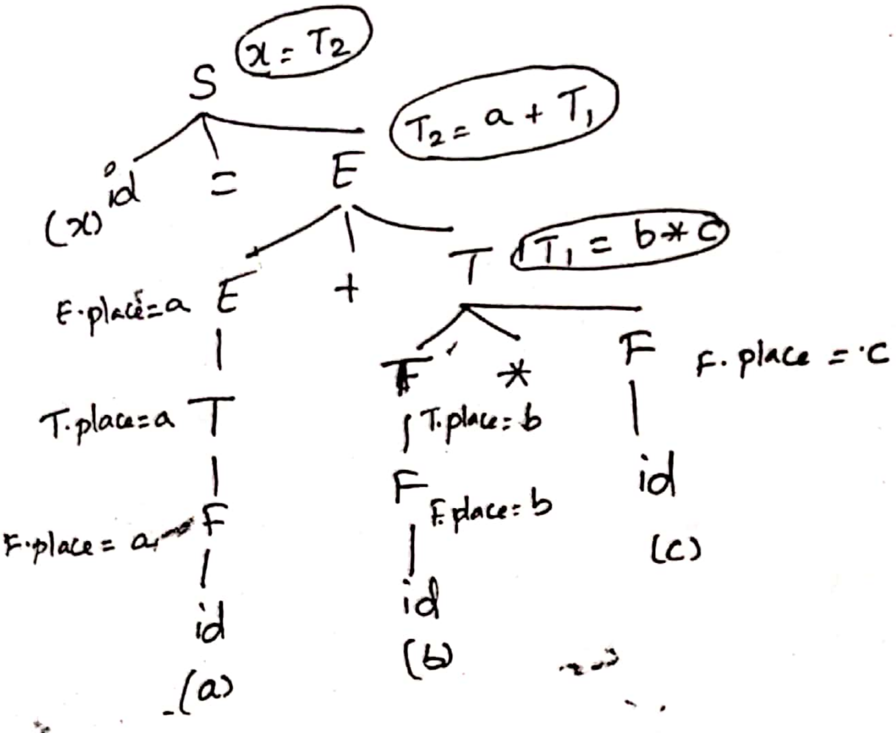


SDT for three address code generation.

```
S → id = E      { gen (id.name = E place) ;
E → E1 + T |    { E. place = new Temp() ;
                  gen (E.place = E1 . place + T. place) ; }

    T           { E.place = T.place ; }

T → T1 * F |    { T. place = new Temp() ;
                  gen (T.place = T1 . place * F . place) ; }

    F           { T.place = F.place; }
F → id          { F.place = id.name; }
```

Consider the expression,

$x = a + b * c$


x = T2
S
T2 = a + T1
(x) id
=
E
T1 = b*c
E.place=a E
+
T
F
*
F
F. place = c
T.place=a T
T.place=b
F.place=b
F.place = a F
id
id
id
(c)
(b)
(a)