TRANSLATION SCHEME

Program → D Program | ^

D → Code | Function | Koment

Code → Statement Code | If Code | While Code | ^

Statement → **Stmt** *koment*
Stmt → **Variable | Input | Output | Return | Chalao**

Function → *kaam ID* { Funct.id = Id.lex} @ **FuncT** (**PL**) *karo Koment* **Code** *kaam khatam Koment*
FuncT -> *khali* | *adad* { symbolTbl.add(ID.lex,Funct.id, lineNumber}
PL → *ID* { symbolTbl.add(ID)} @ *adad* **MPL** | ^
MPL → **|** *PL* | ^

==============Rakho=======================
Variable → *rakho ID* **Type** { R.id = Id.lex } **R**
Type → @ *adad* | ^
R → := *Val* {
emit(R.id+"="+ Val.v);
R.v =SymbolTable.add(R.id, INT); }
| ^ { R.v = SymbolTable.add(R.id, INT); }

Val → *ID* { Val.v=ID.lex; }
| *Integer* { Val.v=Integer.lex; }
| **Exp** { Val.v=Exp.v; }
| **Chalao** {Val.v=Chalao.v}

================Condition=======================

Condition → C*exp RO* **Cexp** { Condition.V = Exp.ex + Ro.lex + Exp.ex }

// P, S are the recursive vars
E → T P
P → + T P1 | - T P1 | ^
T → F S
S→ % F S | / F S | * F S | ^
F → *ID | Digit*

—---

Actions
NOTE: New temp function will automatically add that variable in symbol table

E →   T   { P.i = T.v }   P   { E.v = P.s ; }

P →   +
       T   {
var =newTemp();
emit(var + "=" + $P.i$ + "+" + T.val);
$P_1.i$ = var;
       P_1 { P.s = $P_1.s$ }

P →   -
       T   {
var =newTemp();
emit(var + "=" + $P.i$ + "-" + T.val);
$P_1.i$ = var
}
       P_1 {P.s = $P_1.s$}

P →   ^   {P.s = P.i i}

T →   F   {Q.i = F.v}
      Q   {T.v = Q.s}

Q →   *

      F   {
var =newTemp();
emit(var + "=" + Q.i + "*" + F.val};
Q_1.i = var
}

        Q_1   {Q.s = Q_1.s}

Q →   /

      F   {
var =newTemp();
emit(var + "=" + Q.i + "/" + F.val};
Q_1.i = var

}

        Q_1   {Q.s = Q_1.s}

Q →   %
      F   {
var =newTemp();
emit(var + "=" + Q.i + "%" + F.val};
Q_1.i = var

}

        Q_1   {Q.s = Q_1.s}

Q →   ^   {Q.s = Q.i}

F → *num*   { F.v = num.lex }

F → *ID*   { F.v = id.lex }

Chalao → *chalao ID* { PLF.i=0; } *(* **PLF** *)* {

var=newTemp();
emit ("call" + ID.lex + PLF.v + "," +var);
Chalao.v = var;
}
PLF → *ID* {
 emit("param "+ ID.lex);
 PLF.i = PLF.i +1;   // +1
 MPLF.i = PLF.i;
 } **MPLF** { PLF.v = MPLF.v; }

PLF → *Integer* {
emit ("param"+Integer.lex);
PLF.i=PLF.i+1;
MPLF.i = PLF.i;
} **MPLF** { PLF.v =MPLF.v; }
PLF → ^ { PLF.v = PLF.i ;}
MPLF → | { PLF.i = MPLF.i ;} **PLF** {MPLF.v = PLF.v;}
MPLF → ^ { MPLF.v = MPLF.i ;}

Koment → Comment | ^

Note: Backpatch has global access to ln, so it patches current line number at the parameter passed to it
IF   → *agar* (**Condition**) *to phir karo* {
lnTrue= n ;
emit ("if" + Condition.v + goto + __) ;
lnFalse= n;
Emit ("goto" + __)
BackPatch(lnTrue)
}
Koment
Code {
IF_end= ln;
emit (goto __)
BackPatch(lnFalse)
}
WG

WP

bas karo

{

BackPatch(IF_end)

BackPatch(WG.val)

}

Koment

WG → *warna agar* **Condition** *to phir* **Koment** {

InTrue_= n;

emit ("if" + Condition.v + goto + __) ;

InFalse_ = n;

emit(goto __)

BackPatch(InTrue_)

}

Code {

WG.v= ln; // storing the current line number for Branch Ending

emit (goto __)

BackPatch(InFalse_)

}

WG → **^**

WP → *warna phir Koment* **Code**

WP → **^**

Return-> wapis bhaijo **Val** { emit ("ret" + Val.v) }

// **Todo** : Add cascading to it

Input → *lo* **InputMsg** >> **ID** { emit("in"+ID.v+"\n") }

InputMsg → **^**

InputMsg → **<< String** { emit ("out" +String .v +"\n") }

Output

Output → *dekhao* << **OutVal** { emit ("out" + OutVal.v +"\n") } **MoreOut**

MoreOut → << **OutVal MoreOut** { emit ("out" + OutVal.v +"\n") }

MoreOut →^

OutVal → *String* { String.lex } | *Val* { Val.v }

Do Until

While → *jab tak* { EvalLine=Ln} (**Condition**) *karo* **Koment**

{

InTrue =n;

Emit (if condition goto __)

}

Code

{ emit("goto" + EvalLine) }

bas karo { BackPatch(InTrue) }

Koment