**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** ( **PL** ) *karo Koment* **Code** *kaam khatam Koment*

**FuncT ->** *khali* | *adad*

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

========================================

==================EExpression=====================

// 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);

P1.i =var;

**P1** { P.s = P1.s }

**P** → -

**T**  {

var =newTemp();

emit(var + “=” + P.i + “-” + T.val);

P1.i =var

}

**P1** {P.s = P1.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};

Q1.i =var

}

**Q1** {Q.s = Q1.s}

**Q** → /

**F** {

var =newTemp();

emit(var + “=” + Q.i + “/” + F.val};

Q1.i =var

}

**Q1** {Q.s = Q1.s}

**Q** → %

**F** {

var =newTemp();

emit(var + “=” + Q.i + “%” + F.val};

Q1.i =var

}

**Q1** {Q.s = Q1.s}

**Q** → ^ {Q.s = Q.i}

**F** → *num*  { F.v = num.lex }

**F** → *ID*  { F.v = id.lex }

=================Function Call======================

**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 | ^**

========================================

**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** {

lnTrue\_= n;

emit ( “if” + Condition.v + goto + \_\_ ) ;

lnFalse\_ = n;

emit( goto \_\_)

BackPatch(lnTrue\_)

}

**Code** {

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

emit (goto \_\_ )

BackPatch(lnFalse\_)

}

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

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

**MoreOut** **→**^

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

Note**:**  Backpatch has global access to ln, so it patches current line number at the parameter passed to it

**While →**  *jab tak* ( **Condition** )  *karo* **Koment**

{

lnTrue = n ;

emit (“if” + Condition.Value goto \_\_ );

lnFalse = n;

Emit (goto \_\_ )

BackPatch(lnTrue)}

**Code**

{ emit( “goto” + lnTrue) }

*bas karo* { BackPatch(lnFalse) }

**Koment**