**Language Description for CMM Lexical Analyser:**

|  |  |  |
| --- | --- | --- |
| **Token** | **Transition Diagram** | **Regular Definition** |
| Variable Name |  | R -> letter (letter ll number)\* |
| Arithmetic Operator |  | R-> \* ll + ll / ll - |
| Relational Operator |  | R-> >A ll <A ll =A ll <>  A-> = ll NULL |
| Numeric Constant |  | R-> 0 ll 1 ll 2 ll 3 ll 4 ll … ll 9 |
| Brackets |  | R-> ( ll { ll [ ll ] ll } ll } |
| Comment |  | R-> #A  A->number ll letter ll symbol |
| Multi Line Comment |  | R->##A##  A-> letter ll digit |
| Comma |  | R->, |
| Semi Colon |  | R->; |
| Assignment Operator |  | R-> <- |
| Datatypes |  | R -> int ll char |
| Keywords |  | R->def ll if ll else ll while ll ret ll print ll read |

**Grammar:**

\* P -> P D | ^

\* D -> V | F

\* V -> T id ;

\* T -> int | char

\* F -> def T id ( OPL ) { SL }

\* OPL -> PL | ^

\* PL -> PAR , PL | PAR

\* PAR -> T id

\* SL -> S SL | ^

\* S -> A | SS | I | PR | RT

\* SS -> if ( E ) S | if ( E ) S else S

\* A -> id <- E2 ;

\* E2 -> E2 N

\* I -> while ( E ) S

\* PR -> print L; | print id;

\* RT-> ret L; | ret ID;

\* L-> 'digit' | 'character' | num

\* E->R E'

\* E' -> == R E' | <> R E' | ^

\* R -> N R'

\* R' -> < N R' | > N R' | <= N R' | >= N R' | ^

\* N -> G Q N'

\* N' -> + Q N' | - Q N' | ^

\* Q -> \* QN' | / Q N' | ^

\* G -> id | num | ( E )

**Translation Scheme:**

**Expression**

S -> ID = E [emit(ID.lex, “ = ”, E.n)] --A1

E -> E + T [string tmp = newname()

emit(tmp, “ = “, E.n, “ + ”, T.n)

E.n = tmp] --A2

E -> E \* T [string tmp = newname();

emit(tmp, “ = “, E.n, “ \* ”, T.n);

E.n = tmp] --A3

E -> E / T [string tmp = newname();

emit(tmp, “ = “, E.n, “ / ”, T.n);

E.n = tmp] --A4

E -> E – T [string tmp = newname();

emit(tmp, “ = “, E.n, “ - ”, T.n);

E.n = tmp] --A5

E -> T [E.n = T.n] --A6

T -> ID [T.n = id.lex] --A7

**Selection Statement**

BE->ID1 RO ID2 [BE.t = n // n is line number --A8

emit (“IF ”, ID1.lex, “ ”, RO.lex, “ ”, ID­2.lex,” goto ”)

BE.f = n

emit (“ goto ”)

S-> IF

(BE) [backpatch(BE.t, n)] --A9

S1 [S1.next = n]

ELSE [emit(“ goto ”); backpatch(BE.f, n)] --A10

S2 [backpatch(s.next,n)] --A11

**Loop Statement**

BE->ID1 RO ID2 [BE.t = n // n is line number --A12

emit (“IF ”, ID1.lex, “ ”, RO.lex, “ ”, ID­2.lex,” goto ”)

BE.f = n

emit (“ goto ”)

S-> WHILE

(BE) [backpatch(BE.t, n)] --A13

S[emit( “goto” , BE.t); backpatch(BE.f, n)] --A14