Context Free Grammar For EasyLang:-

V(Non-terminals) = {Start, Outer_Compound_Statement, Compound_Statement , Function_Declaration, Variable_ Declaration, bid, nid, bid1, nid1, Variable_List, Compound_Assignment, Addition, Multiplication, Variable, Loop, Conditional, Condition, Basic_Condition,Variable_List1, Function_Parameters, Function_Call ,FUNCTION, TAKES, RETURNS, RETURN,TYPE, BOOL_TYPE , NUM_TYPE, WHEN, INPUT, OUTPUT, DO, IF, ELSE, FLOAT, INT, BOOL,EMPTY, IN, OUT }

SIGMA (Alphabet Set) = { IDENTIFIER, CONSTANT, STRING, main , if , when , else , do , return, returns, epsilon , takes, int , float, bool , function, (,), { , } , ; , ‘,’, +, - , *, /, = , <, > , !, and , or , : , in , out}
S (Start symbol) = { Start }
PRODUCTIONS =
Start ->
INT main () { Compound_Statement RETURN CONSTANT;} |
Outer_Compound_Statement INT main () { compound statement RETURN CONSTANT; } |
main () { Compound_Statement RETURN CONSTANT;} Outer_Compound_Statement |
Outer_Compound_Statement INT main () { compound statement RETURN CONSTANT; }
Outer_Compound_Statement

Outer_Compound_Statement ->
Function_Declaration |
Function_Declaration Outer_Compound_Statement |
Variable_Declaration |
Variable_Declaration Outer_Compound_Statement

Compound_Statement ->
Loop |
Conditional |
Compound_Assignment |
Outer_Compount_Statement |
RETURN NUM_TYPE; |
RETURN BOOL_TYPE; |
INPUT |
OUTPUT |
Loop Compound_Statement |
Conditional Compound_Statement |
Compound_Assignment Compound_Statement |
Outer_Compount_Statement Compound_Statement |
RETURN NUM_TYPE; Compound_Statement |
RETURN BOOL_TYPE; Compound_Statement |
INPUT Compound_Statement |
OUTPUT Compound_Statement

Function_Declaration ->
IDENTIFIER TAKES (Variable_List) RETURNS TYPE { Compount_Statement }

Compound_Assignment ->
NUM_TYPE IDENTIFIER = Addition; |
BOOL_TYPE IDENTIFIER = Conditional; |

IDENTIFIER = Addition; |
IDENTIFIER = Conditional ;

Addition ->
Multiplication |
Addition + Multiplication |
Addition – Multiplication

Multiplication ->
Variable |
Multiplication * Variable |
Multiplication / Variable

Variable ->
IDENTIFIER |
CONSTANT |
-IDENTIFIER |
-CONSTANT |
(Addition) |
Function_Call

Loop ->
WHEN (Condition) DO { Compound_Statement }

Conditional ->
IF (Condition) { Compound_Statement } |
IF (Condition){ Compound_Statement } ELSE {Compound_statements } |
IF (Condition){ Compound_Statement } ELSE Conditional

Variable_Declaration ->
NUM_TYPE nid; |
BOOL_TYPE bid;

nid ->
IDENTIFIER |
IDENTIFIER, nid |
IDENTIFIER = Addition |
IDENTIFIER = Addition, nid

bid ->
IDENTIFIER |
IDENTIFIER, bid |
IDENTIFIER = Conditional |
IDENTIFIER = Conditional, bid

nid1 -> IDENTIFIER |
IDENTIFIER = Addition

bid1 -> IDENTIFIER |
IDENTIFIER = Conditional

```
Variable_List ->
EMPTY |
Variable_List1
Variable_List1 = NUM_TYPE nid1 |
BOOL_TYPE bid1 |
NUM_TYPE nid1 , Variable_List1 |
BOOL_TYPE bid1 , Variable_List1

Condition ->
Basic_Condition |
Condition and Basic_Condition |
Condition or Basic_Condition |
!Condition

Basic_Condition ->
IDENTIFIER Relational_Operator Addition |
(Condition)

Relational_Operator ->
>= |
== |
<= |
<  |
>

Function_Parameters ->
nid |
bid |
CONSTANT |
nid, Function_Parameters |
bid, Function_Parameters |
CONSTANT, Function_Parameters |
EMPTY

Function_Call ->
IDENTIFIER (Function_Parameters);

INPUT  ->
IN: IDENTIFIER;

OUTPUT ->
OUT: PRINT;
PRINT ->
IDENTIFIER |
STRING |
IDENTIFIER PRINT |
STRING PRINT

IF -> if

WHEN -> when
```

ELSE -> else

DO -> do

RETURN -> return

NUM_TYPE ->
INT|
FLOAT

BOOL_TYPE -> BOOL

EMPTY -> epsilon

TAKES -> takes

TYPE ->
NUM_TYPE |
BOOL_TYPE

INT -> int

FLOAT -> float

BOOL -> bool

FUNCTION -> function

RETURNS -> returns

IN -> in

OUT -> out

LEX CODE -

```
%{
#define KEYWORD 1
#define IDENTIFIER 2
#define STRING 3
#define CONSTANT 4
#define OPERATOR 5
#define SPECIAL 6
%}
%%
[ \t]+    ;
main |
if |
when |
else |
do |
return |
returns |
epsilon |
takes |
int |
float |
bool |
function |
in |
out                         {return KEYWORD;}
\+ |
\- |
\* |
\/ |
= |
\< |
\> |
! |
and |
or                          {return OPERATOR;}
[0-9]+ |
[0-9]+\.[0-9]+ |
\.[0-9]+                    {return CONSTANT;}
\"[^\"\n]*\"                {return STRING; }
\( |
\) |
\{ |
\} |
; |
, |
:                           {return SPECIAL ;}
[a-zA-Z][a-zA-Z0-9]*        {return IDENTIFIER;}
\n                          {return '\n';}
%%

int yywrap(void){
        return 1;
}
#include<stdio.h>
main(argc,argv)
int argc;
char *argv[];
{
int val;

while(val = yylex()){
         switch(val){
                 case 1: printf("<%s,Keyword> ",yytext);
                         break;
```

```
                    case 5: printf("<%s,Operator> ",yytext);
                              break;
                    case 4: printf("<%s,Constant> ",yytext);
                              break;
                    case 3: printf("<%s,String> ",yytext);
                              break;
                    case 6: printf("<%s,Special> ",yytext);
                              break;
                    case 2: printf("<%s,Indentifier> ",yytext);
                              break;
                    case 10: printf("\n");
          }
}
}
```