Compilers Project Report

Team: 14

Name	Sec	BN
عمرو أحمد محمد	2	7
محمد ابراهيم موسى	2	15
محمود عبدالحميد على	2	22
هيثم محمد عبدالكريم	2	34

Delivered to:

Eng/ Omar Samir

Project Overview:

A simple compiler that use Lex and Yacc programming languages where you can declare variables, constant and use Mathematical and logical expressions, Assignment statement, If statement, IF-then-else statement, while loops, Do while loops, repeat-unit loops ,for loops ,switch statement ,functions

Tools and Technologies used:

1) Flex

2) Bison

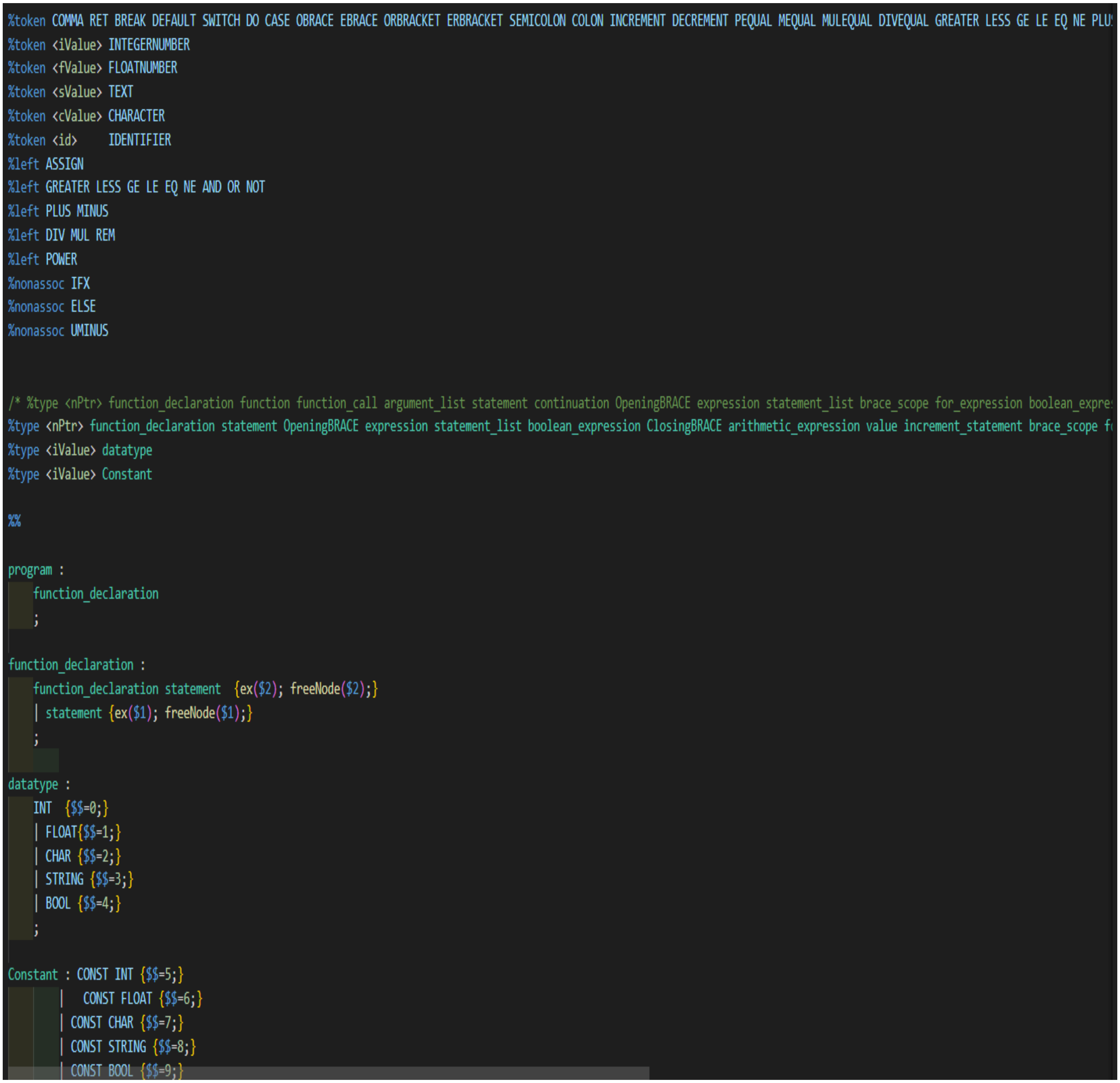
3) Visual studio C#

List of tokens and a description of each:

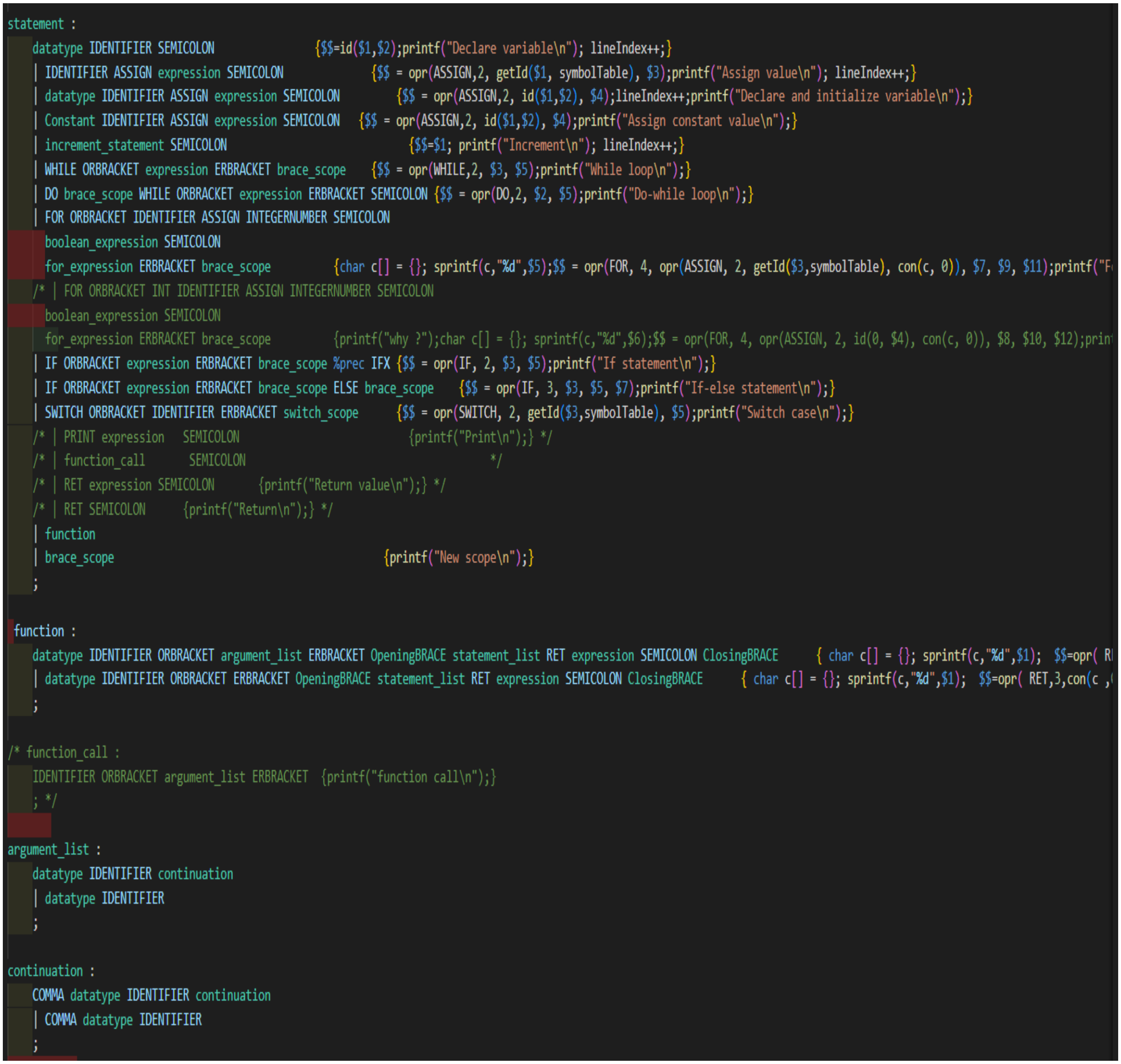
Tokens	Description
{	OBRACE
}	EBRACE
(	ORBRACKET
)	ERBRACKET
;	SEMICOLON
:	COLON
,	COMMA
++	INCREMENT
--	DECREMENT
+=	PLUS EQUAL
-=	MINUS EQUAL
*=	MULTIPLY EQUAL
/=	DIVIDE EQUAL
>	GREATER THAN
<	LESS THAN
>=	GREATER THAN OR EQUAL

<=	LESS THAN OR EQUAL
==	EQUILTY CONDITION
!=	NOT EQUAL
+	PLUS
/	MINUS
*	MULTIPLY
/	DIVIDE
^	POWER
=	ASSIGN
%	Get REMAINDER
&&	AND
||	OR
!	NOT
while	WHILE LOOP
For	FOR LOOP
if	IF CONDITION
else	ELSE
Print	PRINT
Bool	BOOLEAN
Int	INTERGER
Float	FLOAT
Double	DOUBLE
Long	LONG
Char	CHAR
String	STRING
Const	CONSTANT
Do	DO
Break	BREAK
Switch	SWITCH
Case	CASE
False	FALSE
True	TRUE
Default	DEFAULT
Return	RETURN

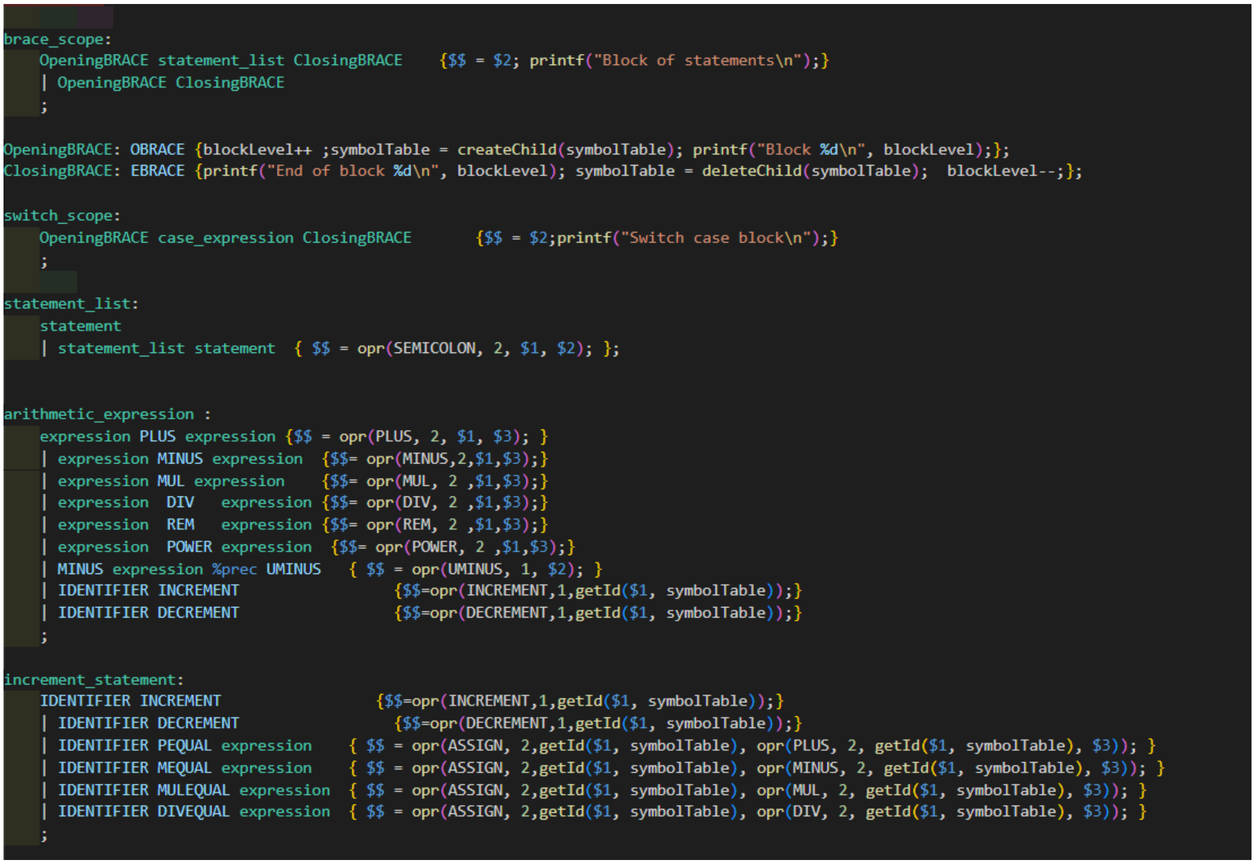
List of language production rules:



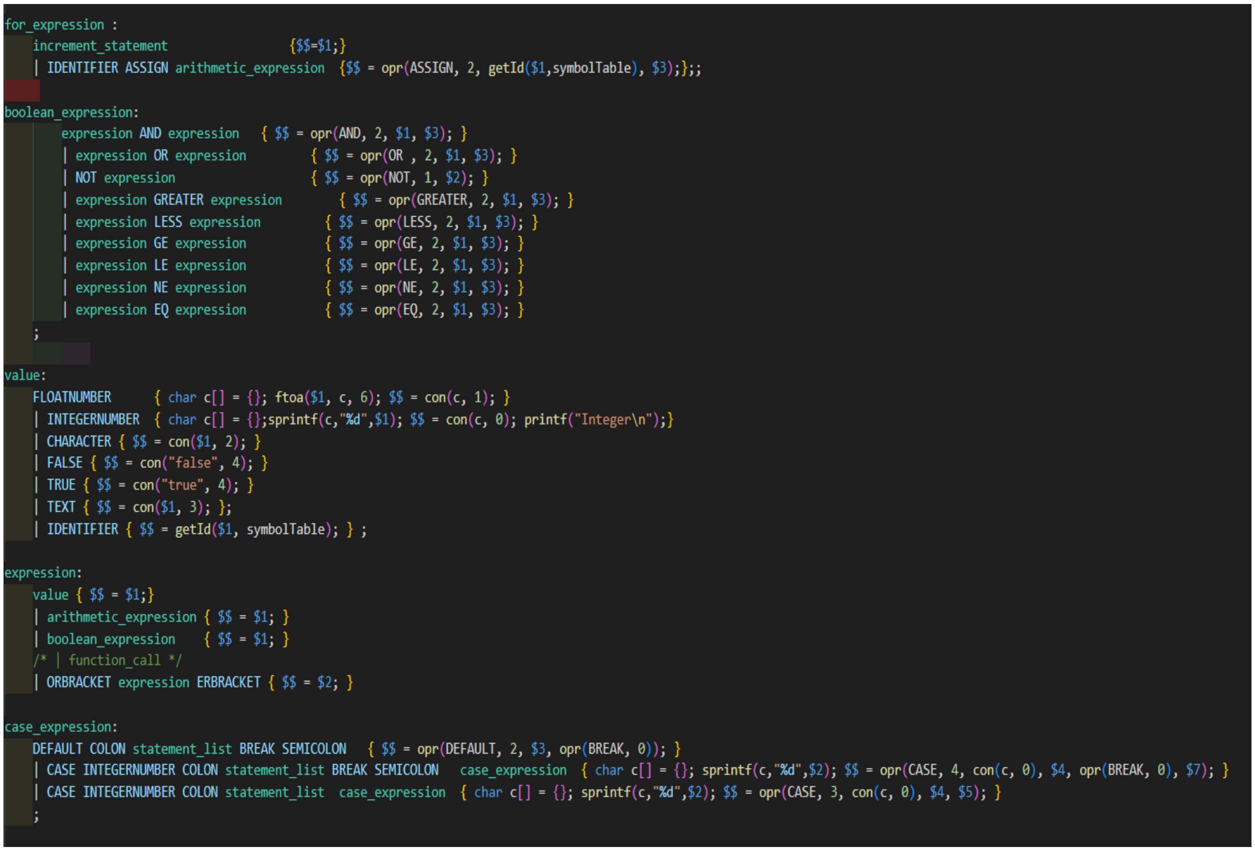
```
%token COMMA RET BREAK DEFAULT SWITCH DO CASE OBRACE EBRACE ORBRACKET ERBRACKET SEMICOLON COLON INCREMENT DECREMENT PEQUAL MEQUAL MULEQUAL DIVEQUAL GREATER LESS GE LE EQ NE PLU
%token <iValue> INTEGERNUMBER
%token <fValue> FLOATNUMBER
%token <sValue> TEXT
%token <cValue> CHARACTER
%token <id>     IDENTIFIER
%left ASSIGN
%left GREATER LESS GE LE EQ NE AND OR NOT
%left PLUS MINUS
%left DIV MUL REM
%left POWER
%nonassoc IFX
%nonassoc ELSE
%nonassoc UMINUS


/* %type <nPtr> function_declaration function function_call argument_list statement continuation OpeningBRACE expression statement_list brace_scope for_expression boolean_expre
%type <nPtr> function_declaration statement OpeningBRACE expression statement_list boolean_expression ClosingBRACE arithmetic_expression value increment_statement brace_scope f
%type <iValue> datatype
%type <iValue> Constant

%%

program :
    function_declaration
    ;

function_declaration :
    function_declaration statement  {ex($2); freeNode($2);}
    | statement {ex($1); freeNode($1);}
    ;

datatype :
    INT  {$$=0;}
    | FLOAT{$$=1;}
    | CHAR {$$=2;}
    | STRING {$$=3;}
    | BOOL {$$=4;}
    ;

Constant : CONST INT {$$=5;}
        |   CONST FLOAT {$$=6;}
        | CONST CHAR {$$=7;}
        | CONST STRING {$$=8;}
        | CONST BOOL {$$=9;}
```



```
statement :
    datatype IDENTIFIER SEMICOLON                {$$=id($1,$2);printf("Declare variable\n"); lineIndex++;}
    | IDENTIFIER ASSIGN expression SEMICOLON             {$$ = opr(ASSIGN,2, getId($1, symbolTable), $3);printf("Assign value\n"); lineIndex++;}
    | datatype IDENTIFIER ASSIGN expression SEMICOLON        {$$ = opr(ASSIGN,2, id($1,$2), $4);lineIndex++;printf("Declare and initialize variable\n");}
    | Constant IDENTIFIER ASSIGN expression SEMICOLON   {$$ = opr(ASSIGN,2, id($1,$2), $4);printf("Assign constant value\n");}
    | increment_statement SEMICOLON                          {$$=$1; printf("Increment\n"); lineIndex++;}
    | WHILE ORBRACKET expression ERBRACKET brace_scope    {$$ = opr(WHILE,2, $3, $5);printf("While loop\n");}
    | DO brace_scope WHILE ORBRACKET expression ERBRACKET SEMICOLON {$$ = opr(DO,2, $2, $5);printf("Do-while loop\n");}
    | FOR ORBRACKET IDENTIFIER ASSIGN INTEGERNUMBER SEMICOLON
      boolean_expression SEMICOLON
      for_expression ERBRACKET brace_scope          {char c[] = {}; sprintf(c,"%d",$5);$$ = opr(FOR, 4, opr(ASSIGN, 2, getId($3,symbolTable), con(c, 0)), $7, $9, $11);printf("F
    /* | FOR ORBRACKET INT IDENTIFIER ASSIGN INTEGERNUMBER SEMICOLON
      boolean_expression SEMICOLON
      for_expression ERBRACKET brace_scope          {printf("why ?");char c[] = {}; sprintf(c,"%d",$6);$$ = opr(FOR, 4, opr(ASSIGN, 2, id(0, $4), con(c, 0)), $8, $10, $12);prin
    | IF ORBRACKET expression ERBRACKET brace_scope %prec IFX {$$ = opr(IF, 2, $3, $5);printf("If statement\n");}
    | IF ORBRACKET expression ERBRACKET brace_scope ELSE brace_scope    {$$ = opr(IF, 3, $3, $5, $7);printf("If-else statement\n");}
    | SWITCH ORBRACKET IDENTIFIER ERBRACKET switch_scope      {$$ = opr(SWITCH, 2, getId($3,symbolTable), $5);printf("Switch case\n");}
    /* | PRINT expression   SEMICOLON                          {printf("Print\n");} */
    /* | function_call       SEMICOLON                                      */
    /* | RET expression SEMICOLON       {printf("Return value\n");} */
    /* | RET SEMICOLON      {printf("Return\n");} */
    | function
    | brace_scope                                         {printf("New scope\n");}
    ;

function :
    datatype IDENTIFIER ORBRACKET argument_list ERBRACKET OpeningBRACE statement_list RET expression SEMICOLON ClosingBRACE      { char c[] = {}; sprintf(c,"%d",$1);  $$=opr( R
    | datatype IDENTIFIER ORBRACKET ERBRACKET OpeningBRACE statement_list RET expression SEMICOLON ClosingBRACE      { char c[] = {}; sprintf(c,"%d",$1);  $$=opr( RET,3,con(c ,
    ;

/* function_call :
    IDENTIFIER ORBRACKET argument_list ERBRACKET  {printf("function call\n");}
    ; */

argument_list :
    datatype IDENTIFIER continuation
    | datatype IDENTIFIER
    ;

continuation :
    COMMA datatype IDENTIFIER continuation
    | COMMA datatype IDENTIFIER
    ;
```



```
brace_scope:
    OpeningBRACE statement_list ClosingBRACE    {$$ = $2; printf("Block of statements\n");}
    | OpeningBRACE ClosingBRACE
    ;

OpeningBRACE: OBRACE {blockLevel++ ;symbolTable = createChild(symbolTable); printf("Block %d\n", blockLevel);};
ClosingBRACE: EBRACE {printf("End of block %d\n", blockLevel); symbolTable = deleteChild(symbolTable);  blockLevel--;};

switch_scope:
    OpeningBRACE case_expression ClosingBRACE       {$$ = $2;printf("Switch case block\n");}
    ;

statement_list:
    statement
    | statement_list statement  { $$ = opr(SEMICOLON, 2, $1, $2); };


arithmetic_expression :
    expression PLUS expression {$$ = opr(PLUS, 2, $1, $3); }
    | expression MINUS expression  {$$= opr(MINUS,2,$1,$3);}
    | expression MUL expression    {$$= opr(MUL, 2 ,$1,$3);}
    | expression  DIV   expression {$$= opr(DIV, 2 ,$1,$3);}
    | expression  REM   expression {$$= opr(REM, 2 ,$1,$3);}
    | expression  POWER expression  {$$= opr(POWER, 2 ,$1,$3);}
    | MINUS expression %prec UMINUS   { $$ = opr(UMINUS, 1, $2); }
    | IDENTIFIER INCREMENT                  {$$=opr(INCREMENT,1,getId($1, symbolTable));}
    | IDENTIFIER DECREMENT                  {$$=opr(DECREMENT,1,getId($1, symbolTable));}
    ;

increment_statement:
    IDENTIFIER INCREMENT                  {$$=opr(INCREMENT,1,getId($1, symbolTable));}
    | IDENTIFIER DECREMENT                  {$$=opr(DECREMENT,1,getId($1, symbolTable));}
    | IDENTIFIER PEQUAL expression    { $$ = opr(ASSIGN, 2,getId($1, symbolTable), opr(PLUS, 2, getId($1, symbolTable), $3)); }
    | IDENTIFIER MEQUAL expression    { $$ = opr(ASSIGN, 2,getId($1, symbolTable), opr(MINUS, 2, getId($1, symbolTable), $3)); }
    | IDENTIFIER MULEQUAL expression  { $$ = opr(ASSIGN, 2,getId($1, symbolTable), opr(MUL, 2, getId($1, symbolTable), $3)); }
    | IDENTIFIER DIVEQUAL expression  { $$ = opr(ASSIGN, 2,getId($1, symbolTable), opr(DIV, 2, getId($1, symbolTable), $3)); }
    ;
```



```
for_expression :
    increment_statement                 {$$=$1;}
    | IDENTIFIER ASSIGN arithmetic_expression  {$$ = opr(ASSIGN, 2, getId($1,symbolTable), $3);};;

boolean_expression:
        expression AND expression    { $$ = opr(AND, 2, $1, $3); }
        | expression OR expression         { $$ = opr(OR , 2, $1, $3); }
        | NOT expression                   { $$ = opr(NOT, 1, $2); }
        | expression GREATER expression        { $$ = opr(GREATER, 2, $1, $3); }
        | expression LESS expression         { $$ = opr(LESS, 2, $1, $3); }
        | expression GE expression           { $$ = opr(GE, 2, $1, $3); }
        | expression LE expression           { $$ = opr(LE, 2, $1, $3); }
        | expression NE expression           { $$ = opr(NE, 2, $1, $3); }
        | expression EQ expression           { $$ = opr(EQ, 2, $1, $3); }
    ;

value:
    FLOATNUMBER      { char c[] = {}; ftoa($1, c, 6); $$ = con(c, 1); }
    | INTEGERNUMBER  { char c[] = {};sprintf(c,"%d",$1); $$ = con(c, 0); printf("Integer\n");}
    | CHARACTER { $$ = con($1, 2); }
    | FALSE { $$ = con("false", 4); }
    | TRUE { $$ = con("true", 4); }
    | TEXT { $$ = con($1, 3); };
    | IDENTIFIER { $$ = getId($1, symbolTable); } ;

expression:
    value { $$ = $1;}
    | arithmetic_expression { $$ = $1; }
    | boolean_expression    { $$ = $1; }
    /* | function_call */
    | ORBRACKET expression ERBRACKET { $$ = $2; }

case_expression:
    DEFAULT COLON statement_list BREAK SEMICOLON   { $$ = opr(DEFAULT, 2, $3, opr(BREAK, 0)); }
    | CASE INTEGERNUMBER COLON statement_list BREAK SEMICOLON   case_expression  { char c[] = {}; sprintf(c,"%d",$2); $$ = opr(CASE, 4, con(c, 0), $4, opr(BREAK, 0), $7); }
    | CASE INTEGERNUMBER COLON statement_list  case_expression  { char c[] = {}; sprintf(c,"%d",$2); $$ = opr(CASE, 3, con(c, 0), $4, $5); }
    ;
```

List of the quadruples and a shot description of each:

Quadruple	Description
Mov R0, v	Move value to R0
Add R2, R1, R0	R2= R1+R0
mul R2, R1, R0	R2= R*R0
Inc R1	Value in R1++
Dec R0	Value in R0--
compGREATER R3, R2, R1	Compare if R1 > R2 and result in R3
compLESS R3, R2, R1	Compare if R1 < R2 and result in R3
compGE R3, R2, R1	Compare if R1 >= R2 and result in R3
compLE R3, R2, R1	Compare if R1 <= R2 and result in R3
compNE R3, R2, R1	Compare if R1 != R2 and result in R3
compEQ R3, R2, R1	Compare if R1 == R2 and result in R3
Jnz L	Jump to label L if not equal zero
Jmp Label	Unconditional jump to Label L