lab9.lxi

```
%{
#include "lab9.tab.h"
%}
%option noyywrap
%option caseless

DIGIT [0-9]
NZD [1-9]
LETTER [a-zA-Z]
IDENTIFIER [a-zA-Z][a-zA-Z0-9_]*
UNDERLINE _
STRING ["][^\n]*["]
BOOLEAN 1|0
NUMBER (\+?|-){NZD}{DIGIT}*|0

%%

[\t\n ]+

"<=" {return LE;}
">=" {return GE;}
"==" {return EQUAL;}
"!=" {return NE;}
"<" {return LESS;}
">" {return GREATER;}
or {return OR;}
and {return AND;}
```

```
"}" {return BR_CURLY_CLOSED;}
"{" {return BR_CURLY_OPENED;}
"[" {return BR_SQUARE_OPENED;}
"]" {return BR_SQUARE_CLOSED;}
"(" {return BR_ROUND_OPENED;}
")" {return BR_ROUND_CLOSED;}
";" {return DOT_COMMA;}
":" {return DOT_DOT;}
"." {return DOT;}
"," {return COMMA;}
"+" {return PLUS;}
"-" {return MINUS;}
"*" {return MULTIPLY;}
"/" {return DIVIDE;}
"%" {return MOD;}
"=" {return ASSIGN;}

if {return IF;}
string {return STRING;}
int {return INT;}
bool {return BOOL;}
array {return ARRAY;}
else {return ELSE;}
while {return WHILE;}
read {return READ;}
write {return WRITE;}

{IDENTIFIER} {return ID;}

{STRING} {return CONST_STRING;}
```

```
{BOOLEAN} {return CONST_BOOLEAN;}
{NUMBER} {return CONST_INT;}
```

lab9.y

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define YYDEBUG 1

int yylex();
void yyerror();

int prodString_len = 0;
int productions[200];

void insertToProdString(int nrProd) {
    productions[prodString_len++] = nrProd;
}

void printProdString(){
    int i;
    printf("Production string: \n");
    for (i = 0; i < prodString_len; i++){
        printf("%d ", productions[i]);
    }
    printf("\n");
}
%}

%token LE
%token GE
%token EQUAL
%token NE
%token OR
%token AND

%token IF
%token STRING
%token INT
%token BOOL
%token ARRAY
%token ELSE
%token WHILE
%token READ
%token WRITE

%token ID

%token CONST_STRING
%token CONST_BOOLEAN
%token CONST_INT

%token BR_CURLY_OPENED
%token BR_CURLY_CLOSED
%token BR_SQUARE_OPENED
%token BR_SQUARE_CLOSED
%token BR_ROUND_OPENED
```

```
%token BR_ROUND_CLOSED

%token DOT_COMMA
%token DOT_DOT
%token DOT
%token COMMA
%token PLUS
%token MINUS
%token MULTIPLY
%token DIVIDE
%token MOD
%token LESS
%token GREATER
%token ASSIGN

%left '+' '-'
%left '*' '/' '%'
%left OR
%left AND

%%
program: BR_CURLY_OPENED stmtlist BR_CURLY_CLOSED {insertToProdString(1);};

type:   primitive {insertToProdString(2);}
    | arraydeclr {insertToProdString(3);};

primitive:  INT {insertToProdString(4);}
            | BOOL {insertToProdString(5);}
            | STRING {insertToProdString(6);};
```

```
arraydeclr: ARRAY BR_SQUARE_OPENED CONST_INT BR_SQUARE_CLOSED
primitive {insertToProdString(7);};


stmtlist: stmt {insertToProdString(8);}
        | stmt stmtlist {insertToProdString(9);};


stmt: simplStmt {insertToProdString(10);}
    | cmpStmt {insertToProdString(11);}
    | ifStmt {insertToProdString(12);}
    | whileStmt {insertToProdString(13);};


simplStmt: assignStmt DOT_COMMA {insertToProdString(14);}
        | ioStmt DOT_COMMA {insertToProdString(15);}
        | declrStmt DOT_COMMA {insertToProdString(16);};


assignStmt: ID ASSIGN expr DOT_COMMA {insertToProdString(17);};


declrStmt: ID DOT_DOT type DOT_COMMA {insertToProdString(18);};


expr: ID {insertToProdString(19);}
    | ID arithmetic_operator ID {insertToProdString(20);};


arithmetic_operator: PLUS {insertToProdString(21);}
                    | MINUS {insertToProdString(22);}
                    | DIVIDE {insertToProdString(23);}
                    | MULTIPLY {insertToProdString(24);};


ioStmt: READ BR_ROUND_OPENED ID BR_ROUND_CLOSED DOT_COMMA
{insertToProdString(25);}
        | WRITE BR_ROUND_OPENED ID BR_ROUND_CLOSED DOT_COMMA
{insertToProdString(26);};
```

```
cmpStmt: BR_CURLY_OPENED stmtlist BR_CURLY_CLOSED
{insertToProdString(27);};


ifStmt: IF BR_ROUND_OPENED cond BR_ROUND_CLOSED BR_CURLY_OPENED
stmt BR_CURLY_CLOSED {insertToProdString(28);}

        | IF BR_ROUND_OPENED cond BR_ROUND_CLOSED BR_CURLY_OPENED stmt
BR_CURLY_CLOSED ELSE BR_CURLY_OPENED stmt BR_CURLY_CLOSED
{insertToProdString(29);};


whileStmt: WHILE BR_ROUND_OPENED cond BR_ROUND_CLOSED
BR_CURLY_OPENED stmt BR_CURLY_CLOSED {insertToProdString(30);};


logical_operator:  {insertToProdString(31);}

                | OR {insertToProdString(32);};


cond: expr relation expr {insertToProdString(33);}

    | expr relation expr logical_operator cond {insertToProdString(34);};


relation: LESS {insertToProdString(35);}

        | LE {insertToProdString(36);}

        | GREATER {insertToProdString(37);}

        | GE {insertToProdString(38);}

        | NE {insertToProdString(39);}

        | EQUAL {insertToProdString(40);};


%%


void yyerror(char *s){

printf("%s\n", s);

}
```

```
extern FILE *yyin;


int main(int argc, char **argv){


if(argc>1) yyin = fopen(argv[1], "r");


if((argc>2)&&(!strcmp(argv[2],"-d"))) yydebug = 1;


if(!yyparse()) {
   fprintf(stderr,"\tO.K.\n");
   printProdString();
}
}
```