1) **Your well documented (with your name and date) YACC code**

```
/*
  Lab5, LEX and YACC routines using BNF to parse input


  Christian McGovern
  Feb 28 2018
*/




%{      /* begin specs */
#include <stdio.h>
#include <ctype.h>
#include "lex.yy.c"
int base, debugsw;

void yyerror (s)  /* Called by yyparse on error */
     char *s;
{
  printf ("%s on Line number: %d\n", s, lineno);
}



%}//end of c definitions
```

```
/*  defines the start symbol, what values come back from LEX and how the
operators are associated  */


%start P


%union{

        int value;

        char * string;

}



%token INT VOID IF ELSE WHILE RETURN READ WRITE LE LT GT GE EQ
NE

%token <string> ID

%token <value> NUM


%left '|'

%left '&'

%left '+' '-'

%left '*' '/' '%'

%left UMINUS


%%    /* end specs, begin rules */
```

```
P               :       DL /*program -Decleration-list*/

                ;

DL              :       DEC

                |   DEC DL

                ;


DEC     : VARDEC | FUNDEC

                ;


VARDEC  : typespec ID ';'

                | typespec ID '[' NUM ']' ';'

                ;


typespec        : INT

                        | VOID

                        ;


FUNDEC  : typespec ID '(' params ')' compoundstmt

                ;


params          : VOID

                | paramlist
                ;

paramlist       : param
                        | param ',' paramlist
                        ;

param   : typespec ID
                | typespec ID '[' ']'
                ;

compoundstmt : '{' localdeclarations statementlist '}'
                        ;

localdeclarations : /* empty */
                                | VARDEC localdeclarations
                                ;

statementlist : /*empty */
                        | statement statementlist
                        ;

statement :  expressionstmt
                 |       compoundstmt
```

```
                  |  selectionstmt
                  |  iterationstmt
                  |  assignmentstmt
                  |  returnstmt
                  |  readstmt
                  |  writestmt
                  ;

expressionstmt : ';'
                         | expression ';'
                         ;

assignmentstmt  : var '=' expression ';'
                         ;

selectionstmt : IF '(' expression ')' statement
                        | IF '(' expression ')' statement ELSE statement
                        ;

iterationstmt : WHILE '(' expression ')' statement
                       ;

returnstmt : RETURN ';'
```

```
                        | RETURN expression ';'
                        ;

readstmt : READ var ';'
                      ;

writestmt : WRITE expression ';'
                       ;

expression : simpleexpression
                        ;

var : ID
    | ID '[' expression ']'
    ;

simpleexpression : additiveexpression
                                             | additiveexpression relop simpleexpression
                                             ;

relop : LE | LT | GT | GE | EQ | NE
      ;
```

```
additiveexpression : term
                   | term addop additiveexpression
                   ;

addop : '+' | '-'
      ;

term : factor
     | factor multop term
     ;

multop : '*' | '/'
   ;

factor : '(' expression ')' | NUM | var | call
   ;

call : ID '(' args ')'
     ;

args : /*empty*/ | arglist
     ;
```

```
arglist : expression
            | expression ',' arglist
            ;



%% /* end rules */


main()
{ yyparse();
}
```

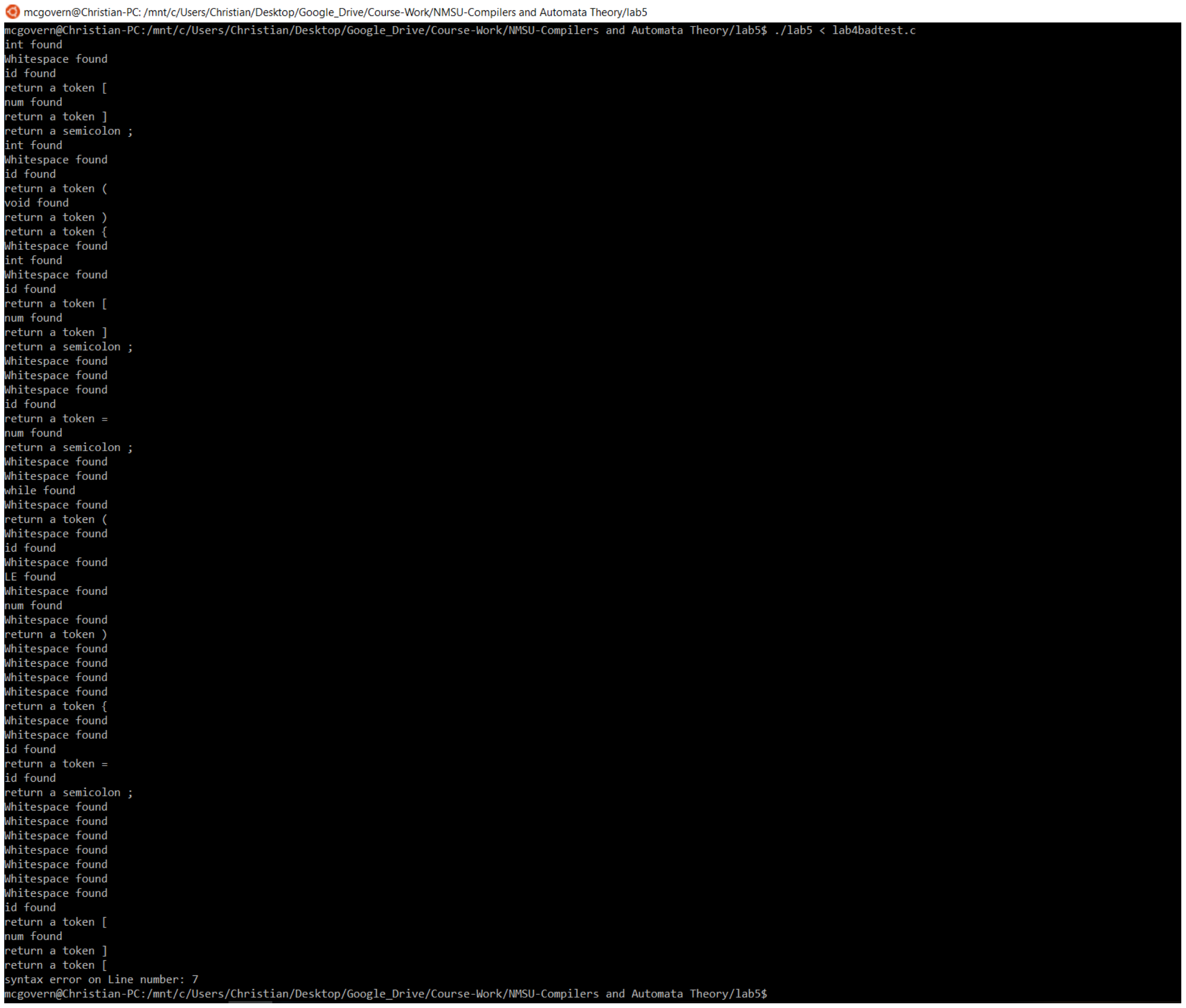
2) **Your well documented LEX code**

```
/*
*  Lex Program
* Christian McGovern
* Lab 5, 2/28
*/
%{


int mydebug=1;
int lineno=1;
#include "y.tab.h"
%}

letter          [a-zA-Z][a-zA-Z]*
num             [0-9][0-9]*
id                      {letter}({letter}|{num})*

/*Begin Rules*/
%%

int             {if (mydebug) fprintf(stderr,"int found\n");
                                                  return(INT);}
void            {if (mydebug) fprintf(stderr,"void found\n");
                                                  return(VOID);}
if                      {if (mydebug) fprintf(stderr,"if found\n");
                                                  return(IF);}
else            {if (mydebug) fprintf(stderr,"else found\n");
                                                  return(ELSE);}
while           {if (mydebug) fprintf(stderr,"while found\n");
                                                  return(WHILE);}
return          {if (mydebug) fprintf(stderr,"return found\n");
                                                  return(RETURN);}
read            {if (mydebug) fprintf(stderr,"read found\n");
                                                  return(READ);}
```

```
write            {if (mydebug) fprintf(stderr,"write found\n");
                                                    return(WRITE);}

[<][\=]                {if (mydebug) fprintf(stderr,"LE found\n");
                                                    return(LE);}

[>][\=]                {if (mydebug) fprintf(stderr,"GE found\n");
                                                    return(GE);}

[\=][\=]               {if (mydebug) fprintf(stderr,"EQ found\n");
                                                    return(EQ);}

[\!][\=]               {if (mydebug) fprintf(stderr,"LE found\n");
                                                    return(NE);}

[<]                    {if (mydebug) fprintf(stderr,"LT found\n");
                                                    return(LT);}

[>]                    {if (mydebug) fprintf(stderr,"GT found\n");
                                                    return(GT);}



{id}             {if (mydebug) fprintf(stderr,"id found\n");
```

```
            yylval.string=strdup(yytext);return(ID);}



{num}        {if (mydebug) fprintf(stderr,"num found\n");

            yylval.value=atoi((const char *)yytext); return(NUM);}





[ \t]        {if (mydebug) fprintf(stderr,"Whitespace found\n");}

[;]              { if (mydebug) fprintf(stderr, "return a semicolon %c\n",
*yytext);

                                   return (*yytext);}

[<>=()\-+*/%&\[\]|;{},]      { if (mydebug) fprintf(stderr,"return a
token %c\n",*yytext); //added () to set

                 return (*yytext);}

\n                        {lineno++;}





%%

/*End Rules*/



int yywrap(void)

{ return 1;}
```

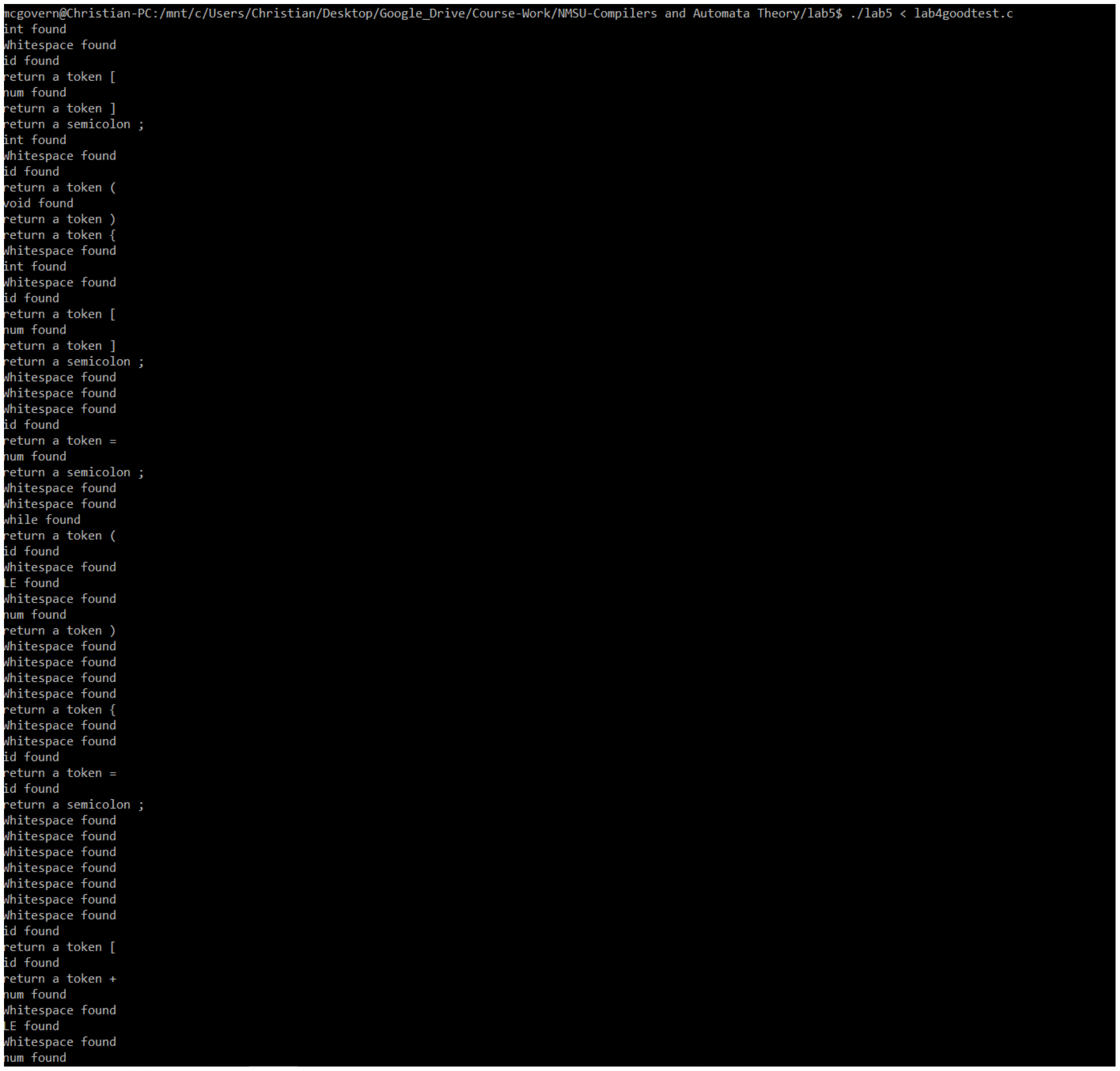
3) **Your output when run with the code lab4badtest.c**



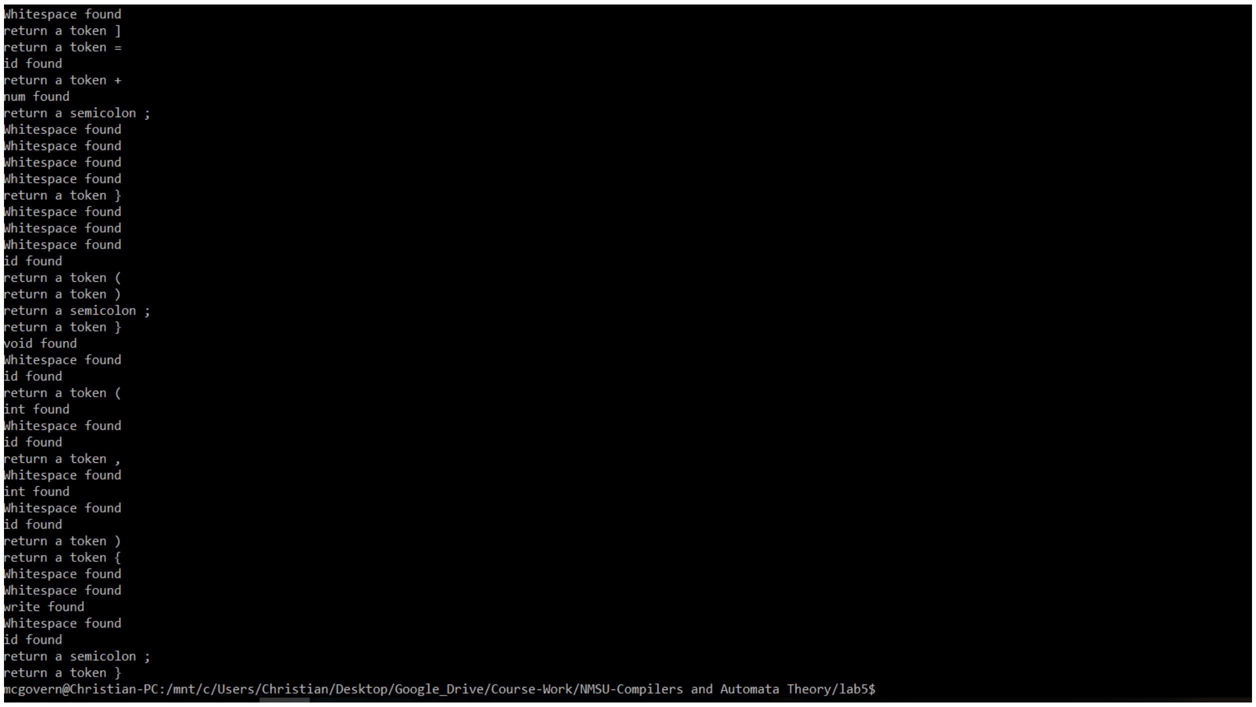
mcgovern@Christian-PC: /mnt/c/Users/Christian/Desktop/Google_Drive/Course-Work/NMSU-Compilers and Automata Theory/lab5

```
mcgovern@Christian-PC:/mnt/c/Users/Christian/Desktop/Google_Drive/Course-Work/NMSU-Compilers and Automata Theory/lab5$ ./lab5 < lab4badtest.c
int found
Whitespace found
id found
return a token [
num found
return a token ]
return a semicolon ;
int found
Whitespace found
id found
return a token (
void found
return a token )
return a token {
Whitespace found
int found
Whitespace found
id found
return a token [
num found
return a token ]
return a semicolon ;
Whitespace found
Whitespace found
Whitespace found
id found
return a token =
num found
return a semicolon ;
Whitespace found
Whitespace found
while found
Whitespace found
return a token (
Whitespace found
id found
Whitespace found
LE found
Whitespace found
num found
Whitespace found
return a token )
Whitespace found
Whitespace found
Whitespace found
Whitespace found
return a token {
Whitespace found
Whitespace found
id found
return a token =
id found
return a semicolon ;
Whitespace found
Whitespace found
Whitespace found
Whitespace found
Whitespace found
Whitespace found
Whitespace found
id found
return a token [
num found
return a token ]
return a token [
syntax error on Line number: 7
mcgovern@Christian-PC:/mnt/c/Users/Christian/Desktop/Google_Drive/Course-Work/NMSU-Compilers and Automata Theory/lab5$
```

4) Your output when run with the code lab4goodtest.c



```
mcgovern@Christian-PC:/mnt/c/Users/Christian/Desktop/Google_Drive/Course-Work/NMSU-Compilers and Automata Theory/lab5$ ./lab5 < lab4goodtest.c
int found
whitespace found
id found
return a token [
num found
return a token ]
return a semicolon ;
int found
whitespace found
id found
return a token (
void found
return a token )
return a token {
whitespace found
int found
whitespace found
id found
return a token [
num found
return a token ]
return a semicolon ;
whitespace found
whitespace found
whitespace found
id found
return a token =
num found
return a semicolon ;
whitespace found
whitespace found
while found
return a token (
id found
whitespace found
LE found
whitespace found
num found
return a token )
whitespace found
whitespace found
whitespace found
whitespace found
return a token {
whitespace found
whitespace found
id found
return a token =
id found
return a semicolon ;
whitespace found
whitespace found
whitespace found
whitespace found
whitespace found
whitespace found
whitespace found
id found
return a token [
id found
return a token +
num found
whitespace found
LE found
whitespace found
num found
```



```
Whitespace found
return a token ]
return a token =
id found
return a token +
num found
return a semicolon ;
Whitespace found
Whitespace found
Whitespace found
Whitespace found
return a token }
Whitespace found
Whitespace found
Whitespace found
id found
return a token (
return a token )
return a semicolon ;
return a token }
void found
Whitespace found
id found
return a token (
int found
Whitespace found
id found
return a token ,
Whitespace found
int found
Whitespace found
id found
return a token )
return a token {
Whitespace found
Whitespace found
write found
Whitespace found
id found
return a semicolon ;
return a token }
mcgovern@Christian-PC:/mnt/c/Users/Christian/Desktop/Google_Drive/Course-Work/NMSU-Compilers and Automata Theory/lab5$
```