*Final Report*

*Compiler design & construction lab*

*Submitted by*

*Md. Yeasir Arafat Rakib ( 14 cse 046 )*

*Course code: cse 4104*

*Submitted To:*

*Ass. Prof. Dr. Md. Manjur Ahmed,*

**

*Department of Computer Science & Engineering,*

**University of Barisal.**

Lab work 1

**Experiment Name**: Designing a lexical analyzer for given language and the lexical analyzer should identify operator, integer and float number.

**Resources**: c/c++ , Ubuntu operating system.

**Program Logic:**

1. Read the input Expression
2. Check whether input is operator or digits then store it as identifier
3. If the input is is operator store it as symbol
4. Check the input for keywords

**Procedure:** Go to debug -> run the program.

**Programes:**

%%

[(]\* printf("open first braket");

[)]\* printf("close first braket ");

[{]\* printf("open second braket ");

[}]\* printf("close second braket ");

[\[]\* printf("open third braket ");

[\]]\* printf("close third braket");

[0-9]+"."[0-9]+ printf("float ");

[1-9][0-9]\*\* printf("integer ");

[a-zA-Z][\_a-zA-Z0-9]\* printf(" identifier");

[\+] printf("plus ");

[\-] printf("minus ");

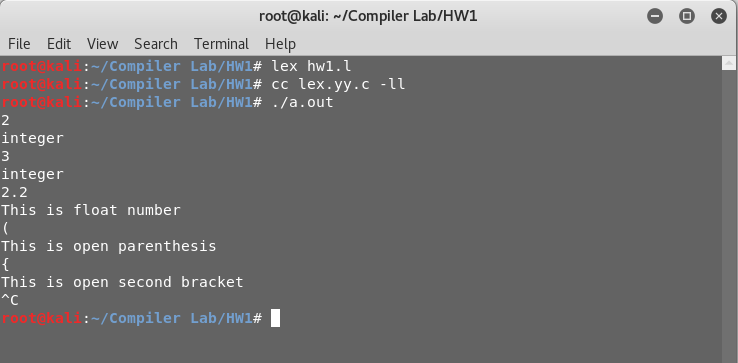
[\\*] printf("multiplication ");

[\%] printf("division ");

. printf("unexpected");

%%

Output:



**Lab work 2**

**Experiment Name**: Designing a lexical analyzer for given language and the lexical analyzer should identify data types, data names or other information keywords..

**Resources**: C/c++ , Ubuntu operating system.

**Program Logic:**

1. Read the input Expression
2. Check whether input is operator or digits or data types then store it as identifier
3. If the input is operator store it as symbol
4. Check the input for keywords

**Programs:**

**myscanner.l**

%{

#include "myscanner.h"

%}

%%

: return COLON;

"db\_type" return TYPE;

"db\_name" return NAME;

"db\_table\_prefix" return TABLE\_PREFIX;

"db\_port" return PORT;

[a-zA-Z][\_a-zA-Z0-9]\* return IDENTIFIER;

[1-9][0-9]\* return INTEGER;

[ \t\n] ;

. printf("unexpected character");

%%

intyywrap(void)

{

return 1;

}

**myscanner.h**

#define TYPE 1

#define NAME 2

#define TABLE\_PREFIX 3

#define PORT 4

#define COLON 5

#define IDENTIFIER 6

#define INTEGER 7

**myscanner.c**

#include <stdio.h>

#include "myscanner.h"

externintyylex();

externintyylineno;

extern char\* yytext;

char \*names[] = {NULL, "db\_type", "db\_name", "db\_table\_prefix", "db\_port"};

int main(void)

{

intntoken, vtoken;

ntoken = yylex();

while(ntoken) {

printf("%d\n", ntoken);

if(yylex() != COLON) {

printf("Syntax error in line %d, Expected a ':' but found %s\n", yylineno, yytext);

return 1;

}

vtoken = yylex();

switch (ntoken) {

case TYPE:

case NAME:

case TABLE\_PREFIX:

if(vtoken != IDENTIFIER) {

printf("Syntax error in line %d, Expected an identifier but found %s\n", yylineno, yytext);

return 1;

}

printf("%s is set to %s\n", names[ntoken], yytext);

break;

case PORT:

if(vtoken != INTEGER) {

printf("Syntax error in line %d, Expected an integer but found %s\n", yylineno, yytext);

return 1;

}

printf("%s is set to %s\n", names[ntoken], yytext);

break;

default:

printf("Syntax error in line %d\n",yylineno);

}

ntoken = yylex();

}

return 0;

}

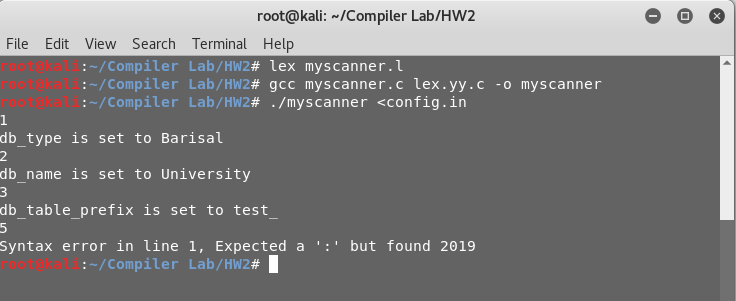
**config.in**

db\_type : sqlite

db\_name : Identity

db\_table\_prefix : data\_option

db\_port: 1111

Output:

**Lab work 3**

**Experiment Name**: Designing a YACC specification for a more advanced desk calculator.

**Resources**: Turbo C , Ubuntu operating system.

**Program Logic:**

1. Read the input Expression
2. Check whether input is operator or digits then store it as identifier
3. If the input is operator store it as symbol
4. Check the input for keywords

**Procedure:** Go to debug -> run or press CTRL + F9 to run the program.

**Code:**

**myCal.l**

delim [ \t]

ws {delim}+

digit [0-9]

number {digit}+

%%

{ws} ;

\n return(NEWLINE);

{number} {(yylval=atoi(yytext));

return(NUMBER);

}

"+" return(PLUS);

"-" return(SUB);

"\*" return(TIMES);

"/" return(DIV);

"(" return(BRA);

")" return(KET);

**myCalparsing.y**

%{

#include <stdio.h>

int yylex();

void yyerror(const char \*s);

%}

%token NUMBER PLUS SUB TIMES DIV BRA KET NEWLINE

%%

lines : line

| line lines

;

line : expr NEWLINE {printf ("value: %d\n",$1);}

;

expr : expr PLUS term {$$=$1+$3;}

| term {$$=$1;}

;

term : term TIMES factor {$$=$1\*$3;}

| factor {$$=$1;}

;

factor: factor SUB factor1 {$$=$1-$3;}

|factor1 {$$=$1;}

;

factor1: factor1 DIV factor2 {$$=$1/$3;}

|factor2 {$$=$1;}

;

factor2 : BRA expr KET {$$=$2;}

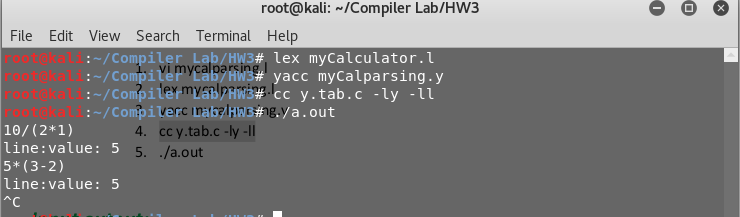
|NUMBER {$$=$1;}

;

%%

#include "lex.yy.c"

**Output:**

****

**LAB 4**

**Hw4.y**

%{

#include<stdio.h>

intyylex();

intsym[26];

voidyyerror(const char \*s);

%}

%left '+' '-'

%left '\*' '/'

%token VARIABLE INTEGER

%%

program :program statement '\n'

|

;

statement :expr { printf("%d\n", $1); }

| VARIABLE '=' expr { sym[$1] = $3; }

;

expr:INTEGER

| VARIABLE { $$ = sym[$1]; }

| expr '+' expr { $$ = $1 + $3; }

| expr '-' expr { $$ = $1 - $3; }

| expr '\*' expr { $$ = $1 \* $3; }

| expr '/' expr { $$ = $1 / $3; }

| '(' expr ')' { $$ = $2; }

;

%%

#include "lex.yy.c"

externintyylex();

externintyyparse();

int main() { yyparse(); }

**hw4.l**

%%

[a-z] {

yylval = \*yytext - 'a';

return VARIABLE;

}

[0-9]+ {

yylval = atoi(yytext);

return INTEGER;

}

[-+()=/\*\n] { return \*yytext; }

[ \t] ;

. yyerror("invalid character");

%%

Output:

