**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**

****

**LAB REPORT**

**on**

**COURSE TITLE**

***Submitted by***

**ARCHIT MEHROTRA (1BM21CS031)**

***in partial fulfillment for the award of the degree of***

**BACHELOR OF ENGINEERING**

***in***

**COMPUTER SCIENCE AND ENGINEERING**



**B.M.S. COLLEGE OF ENGINEERING**

**(Autonomous Institution under VTU)**

**BENGALURU-560019**

**October-2023 to Feb-2024**

**B. M. S. College of Engineering,**

**Bull Temple Road, Bangalore 560019**

(Affiliated To Visvesvaraya Technological University, Belgaum)

**Department of Computer Science and Engineering**



**CERTIFICATE**

This is to certify that the Lab work entitled “Compiler Design” carried out by **ARCHIT MEHROTRA (1BM21CS031),** who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2023. The Lab report has been approved as it satisfies the academic requirements in respect of a **Compiler Design course (21CS5PCCPD)**work prescribed for the said degree.

## Sunayana S               Dr. Jyothi S Nayak

Assistant Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

`

**Index**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl. No.** | **Date** | **Experiment Title** | **Page No.** |
| **01** |  | **Write a program to design Lexical Analyzer in (to recognize any five keywords, identifiers, numbers, operators and punctuations)** |  |
| **02** |  | **Write a program in LEX to recognize Floating Point Numbers.** |  |
| **03** |  | **Write a program in LEX to recognize different tokens: Keywords, Identifiers, Constants, Operators and Punctuation symbols.** |  |
| **04** |  | **Write a LEX program that copies a file, replacing each nonempty sequence of white spaces by a single blank.** |  |
| **05** |  | **Write a LEX program to recognize the following tokens over the alphabets {0,1,..,9}**   1. **The set of all string ending in 00.** 2. **The set of all strings with three consecutive 222’s.** 3. **The set of all string such that every block of five**   **consecutive symbols contains at least two 5’s.**   1. **The set of all strings beginning with a 1 which,**   **interpreted as the binary representation of an**  **integer, is congruent to zero modulo 5.**   1. **The set of all strings such that the 10th symbol**   **from the right end is 1.**   1. **The set of all four digits numbers whose sum is 9** 2. **The set of all four digital numbers, whose** 3. **individual digits are in ascending order from left to**   **right.** |  |
| **06** |  | **Write a program to implement**   1. **Recursive Descent Parsing with back tracking**   **(Brute Force Method). S→ cAd , A →ab /a**   1. **Recursive Descent Parsing with back tracking**   **(Brute Force Method). S→ cAd , A → a / ab** |  |
| **07** |  | **Write a program to implement: Recursive Descent Parsing with back tracking (Brute Force Method).**   1. **S→ aaSaa | aa** 2. **S → aaaSaaa | aa** 3. **S → aaaaSaaaa | aa** 4. **S → aaaSaaa |aSa | aa** |  |
| **08** |  | **Write a program to design LALR parsing using YACC** |  |
| **09** |  | **Use YACC to Convert Binary to Decimal (including fractional numbers)** |  |
| **10** |  | **Use YACC to implement, evaluator for arithmetic expressions (Desktop calculator)** |  |
| **11** |  | **Use YACC to convert: Infix expression to Postfix expression.** |  |
| **12** |  | **Use YACC to generate Syntax tree for a given expression** |  |
| **13** |  | **Use YACC to generate 3-Address code for a given expression** |  |
| **14** |  | **Use YACC to generate the 3-Address code which contains Arrays.** |  |

Experiment No :01

**Aim of the program**

Write a program to design Lexical Analyzer in C/C++/Java/Python Language (to recognize any five keywords, identifiers, numbers, operators and punctuations)

**Program**

def analyze\_input(input\_text):

keywords = ["char", "float", "bool", "int", "for", "break", "continue"]

punctuation = [".", "!", ";", "?"]

operators = ["+", "-", "\*", "/", "%", "="]

keys, ids, nums, ops, punct = 0, 0, 0, 0, 0

for i in input\_text.split():

if i in keywords:

if keys < 5:

print(f'{i} is a keyword!\n')

keys += 1

elif i in punctuation:

if punct < 5:

print(f'{i} is a punctuation!\n')

punct += 1

elif i in operators:

if ops < 5:

print(f'{i} is an operator!\n')

ops += 1

elif i.isnumeric():

if nums < 5:

print(f'{i} is a number!\n')

nums += 1

else:

if ids < 5:

flag = False

if i[0].isalpha() or i[0] == '\_':

flag = True

for j in i[1:]:

if j in operators or j in punctuation:

print(f'{i} is an invalid token!\n')

flag = False

break

if flag:

print(f'{i} is an identifier!\n')

ids += 1

else:

print(f'{i} is an invalid token!\n')

while True:

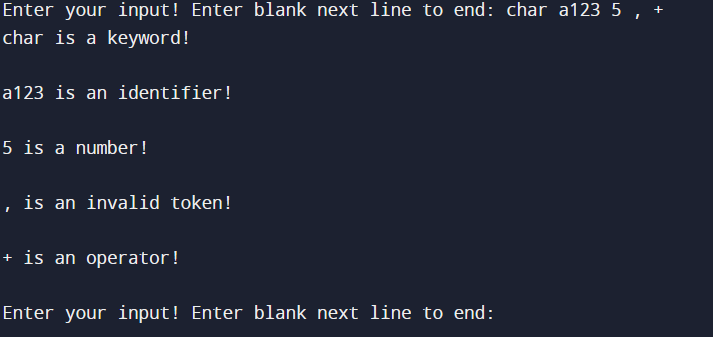
user\_input = input("Enter your input! Enter blank next line to end: ")

if not user\_input.strip():

break

analyze\_input(user\_input)

**Output – Screen shot**



Experiment No :02

**Aim of the program**

Write a program in LEX to recognize Floating Point Numbers.

**Program**

%{

#include<stdio.h>

int flag=0;

%}

alpha[a-zA-Z]

digit[0-9]

decimal[.]

%%

[+|-]?({digit})\*{decimal}({digit})\* { flag=1;}

{alpha}({alpha}|{digit})\* {printf("invalid number ");}

\n return 0;

%%

int yywrap(){}

int main(){

printf("enter :");

yylex();

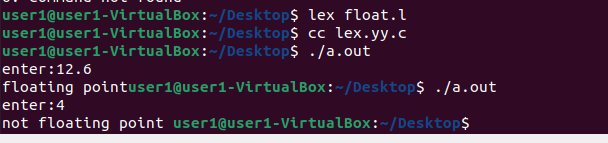
if(flag==1){ printf("floating point number");}

else{printf(" not a floating point number");}

}

}

**Output – Screen shot**

****

Experiment No :03

**Aim of the program**

**Write a program in LEX to recognize different tokens: Keywords, Identifiers, Constants, Operators**

**and Punctuation symbols.**

**Program**

%{

#include<stdio.h>

int x1=0,x2=0,x3=0,x4=0;

%}

alpha[a-zA-Z]

digit[0-9]

d[.]

%%

int|float|char { x1++;}

{digit}+ {x2++;}

[<|>|=|<=|>=|==] {x3++;}

{alpha}({digit}|{alpha})\* {x4++;}

\n {

printf("\nkey:%d",x1);

printf("\nconst:%d",x2);

printf("\noperator:%d",x3);

printf("\nidentifier:%d",x4);

}

%%

int yywrap(){}

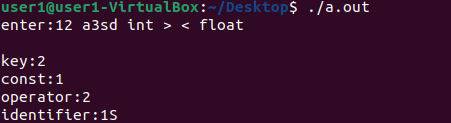
int main(){

printf("enter:");

yylex();

}

**Output – Screen shot**

****

Experiment No :04

**Aim of the program**

**Write a LEX program that copies a file, replacing each nonempty sequence of white spaces by a single blank.**

**Program**

%{

#include<stdio.h>

%}

%%

[ ]([ ])\* {fprintf(yyout," ");}

([ ])\*(\n)([ ])\* {fprintf(yyout," ");}

%%

int yywrap(){}

int main(){

printf("running");

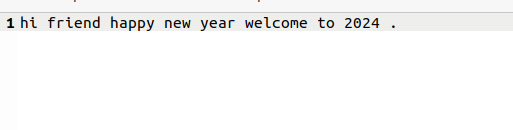
yyin=fopen("txt","r");

yyout=fopen("txto","w");

yylex();

}

**Output – Screen shot**

****

Experiment No :05

**Aim of the program**

**Write a LEX program to recognize the following tokens over the alphabets {0,1,..,9}**

**a) The set of all string ending in 00.**

**b) The set of all strings with three consecutive 222’s.**

**c) The set of all string such that every block of five consecutive symbols contains at least two 5’s.**

**d) The set of all strings beginning with a 1 which, interpreted as the binary representation of an integer, is congruent to zero modulo 5.**

**e) The set of all strings such that the 10th symbol from the right end is 1.**

**f) The set of all four digits numbers whose sum is 9**

**g) The set of all four digital numbers, whose individual digits are in ascending order from left to right.**

**Program**

%{

#include<stdio.h>

int x1=0,x2=0,x3=0,x4=0;

%}

alpha[a-zA-Z]

digit[0-9]

d[.]

%%

({digit})\*00 {printf("\n%s rule A",yytext);}

({digit})\*222({digit})\* {printf("\n%s rule B",yytext);}

(1(0)\*(11|01)(01\*01|00\*10(0)\*(11|1))\*0)(1|10(0)\*(11|01)(01\*01|00\*10(0)\*(11|1))\*10)\* {printf("\n%s rule D",yytext);}

({digit})\*1{digit}{9} {printf("\n%s rule E",yytext);}

{digit}{4} {

int sum=0;

for(int i=0;i<4;i++){

sum=sum+yytext[i]-48;

}

if(sum==9) {printf("\n%s rule F",yytext);}

sum=1;

for(int j=0;j<3;j++){

if(yytext[j]>yytext[j+1]) sum=0;

}

if(sum==1) {printf("\n%s rule G",yytext);}

}

{d}\* {int i=0; int c=0;

if(yyleng<5) {break;}

for(i=0;i<5;i++) {

if(yytext[i]=='5') c++;

}

if(c<2) {break;}

else{

for(;i<yyleng;i++){

if(yytext[i-5]=='5') c--;

if(yytext[i]=='5') c++;

if(c<2) break;

}

if(i==yyleng) {printf("\n %s rule C",yytext);}

}

}

%%

int yywrap(){}

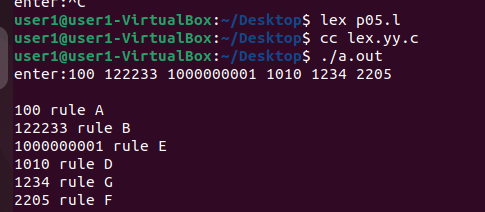
int main(){

printf("enter:");

yylex();

}

**Output – Screen shot**

****

**Part-B:**

Experiment No :01

**Aim of the program**

**1. Write a program to implement**

**(a) Recursive Descent Parsing with back tracking (Brute Force Method). S→ cAd , A →ab /a**

**(b) Recursive Descent Parsing with back tracking (Brute Force Method). S→ cAd , A → a / ab**

**Program**

#include<stdio.h>

#include<conio.h>

#include<string.h>

int A();

char str[15];

int isave,curr\_ptr=0;

int main(void)

{

//clrscr();

printf("1.S->cAd\n2.A->ab/a\n");

printf("this is parser for the above grammar:\n");

printf("Enter any string:");

scanf("%s",str);

while(curr\_ptr<strlen(str))

{

//S has only one immediate derivation which is cAd

//match with c

if (str[curr\_ptr]=='c')

{

curr\_ptr++;

//call function to match A

if (A()) //checking the productions of A->ab/a

{

curr\_ptr++;

//match d

if (str[curr\_ptr]=='d' && str[curr\_ptr+1]=='\0')

{

//success

printf("string is accepted by the grammar");

getch();

return 1;

}

else break;

}

else break;

}

else break;

}

//incase any of them fail to match return negatively.

printf("string is not accepted by the grammar");

//getch();

return 0;

}

int A() //sub function A()

{

isave=curr\_ptr;

if (str[curr\_ptr]=='a')

{

curr\_ptr++;

if(str[curr\_ptr]=='b')

return 1;

}

curr\_ptr=isave; //return to start

//check if a is matched and return accordingly.

if(str[curr\_ptr]=='a')

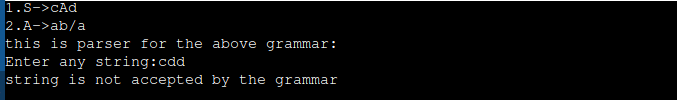
return 1;

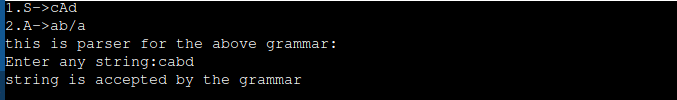
else

return 0;

}

**Output – Screen shot**





Experiment No :02

**Aim of the program**

**Use YACC to Convert Binary to Decimal (including fractional numbers)**

**Program**

**p.y**

%{

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

void yyerror(char \*s);

float x = 0;

%}

%token ZERO ONE POINT

%%

L: X POINT Y {printf("%f",$1+x);}

| X {printf("%d", $$);}

X: X B {$$=$1\*2+$2;}

| B {$$=$1;}

Y: B Y {x=$1\*0.5+x\*0.5;}

| {;}

B:ZERO {$$=$1;}

|ONE {$$=$1;};

%%

int main()

{

printf("Enter the binary number : ");

while(yyparse());

printf("\n");

}

void yyerror(char \*s)

{

fprintf(stdout,"\n%s",s);

}

**p.l**

%{

#include<stdio.h>

#include<stdlib.h>

#include"y.tab.h"

extern int yylval;

%}

%%

0 {yylval=0;return ZERO;}

1 {yylval=1;return ONE;}

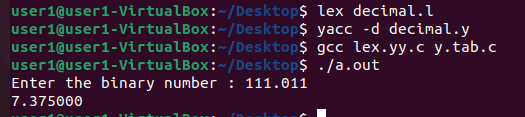
"." {return POINT;}

[ \t] {;}

\n return 0;

%%

**Output – Screen shot**



Experiment No :03

**Aim of the program**

**Use YACC to implement, evaluator for arithmetic expressions (Desktop calculator)**

**Program**

**p.y**

%{

#include<stdio.h>

int flag=0;

int yylex();

int yyerror();

%}

%token NUMBER

%left '+' '-'

%left '\*' '/'

%left '%'

%right '^'

%left '(' ')'

%%

ArithmeticExpression: E{

printf("\nResult=%d\n",$$);

return 0;

}

E:E'+'E {$$=$1+$3;}

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

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

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

|E'%'E {$$=$1%$3;}

|E'^'E {$$=$1^$3;}

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

| NUMBER {$$=$1;}

;

%%

void main()

{

printf("\nEnter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Division, Modulus and Round brackets:\n");

yyparse();

if(flag==0)

printf("\nEntered arithmetic expression is Valid\n\n");

}

int yyerror()

{

printf("\nEntered arithmetic expression is Invalid\n\n");

flag=1;

return 0;

}

**P.l**

%{

#include<stdio.h>

#include "y.tab.h"

extern int yylval;

%}

%%

[0-9]+ {

yylval=atoi(yytext);

return NUMBER;

}

[\t] ;

[\n] return 0;

. return yytext[0];

%%

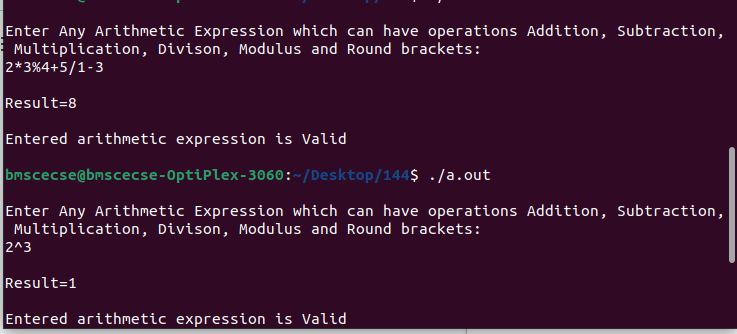
int yywrap()

{

return 1;

}

**Output – Screen shot**



Experiment No :04

**Aim of the program**

**Use YACC to convert: Infix expression to Postfix expression.**

**Program**

**p.y**

%{

#include <ctype.h>

#include<stdio.h>

#include<stdlib.h>

int yylex();

%}

%token digit

%%

S: E {printf("\n\n");}

;

E: E '+' T { printf ("+");}

| E '-' T { printf ("-");}

| T

;

T: T '\*' P { printf("\*");}

| T '/' P { printf("/");}

| P

;

P: F '^' P { printf ("^");}

| F

;

F: '(' E ')'

| digit {printf("%d", $1);}

;

%%

int main()

{

printf("Enter infix expression: ");

yyparse();

}

yyerror()

{

printf("Error");

}

**p.l**

%{

#include "y.tab.h"

extern int yylval;

%}

%%

[0-9]+ {yylval=atoi(yytext); return digit;}

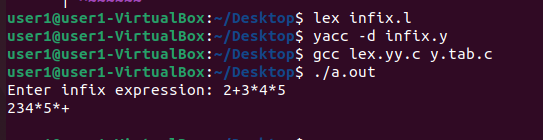
[\t] ;

[\n] return 0;

. return yytext[0];

%%

**Output – Screen shot**

****

Experiment No :05

**Aim of the program**

**Use YACC to generate Syntax tree for a given expression**

**Program**

**p.y**

%{

#include<math.h>

#include<ctype.h>

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include "y.tab.h"

struct tree\_node {

char val[10];

int lc;

int rc;

};

int ind;

struct tree\_node syn\_tree[100];

void my\_print\_tree(int cur\_ind);

int mknode(int lc, int rc, const char \*val);

int yylex(void);

void yyerror(const char \*s);

%}

%token digit

%%

/\* print the tree after evaluating E \*/

S: E { my\_print\_tree($1); }

;

E: E '+' T { $$= mknode($1, $3, "+"); }

| E '-' T { $$= mknode($1, $3, "-"); }

| T { $$= $1; }

;

T: T '\*' F { $$= mknode($1, $3, "\*"); }

| T '/' F { $$= mknode($1, $3, "/"); }

| F { $$= $1; }

;

F: P '^' F { $$= mknode($1, $3, "^"); }

| P { $$= $1; }

;

P: '(' E ')' { $$= $2; }

| digit { char buf[10]; sprintf(buf, "%d", yylval); $$= mknode(-1, -1, buf); }

%%

int main() {

ind=0;

printf("Enter an expression\n");

yyparse();

return 0;

}

void yyerror(const char \*s) {

printf("NITW Error: %s\n", s);

}

int mknode(int lc, int rc, const char \*val) {

strcpy(syn\_tree[ind].val, val);

syn\_tree[ind].lc = lc;

syn\_tree[ind].rc = rc;

ind++;

return ind-1;

}

void my\_print\_tree(int cur\_ind) {

if (cur\_ind == -1) return;

if (syn\_tree[cur\_ind].lc == -1 && syn\_tree[cur\_ind].rc == -1)

printf("Digit Node -> Index: %d, Value: %s\n", cur\_ind, syn\_tree[cur\_ind].val);

else

printf("Operator Node -> Index: %d, Value: %s, Left Child Index: %d, Right Child Index: %d\n",

cur\_ind, syn\_tree[cur\_ind].val, syn\_tree[cur\_ind].lc, syn\_tree[cur\_ind].rc);

my\_print\_tree(syn\_tree[cur\_ind].lc);

my\_print\_tree(syn\_tree[cur\_ind].rc);

}

**p.l**

%{

#include "y.tab.h"

%}

%%

[0-9]+ { yylval=atoi(yytext); return digit; }

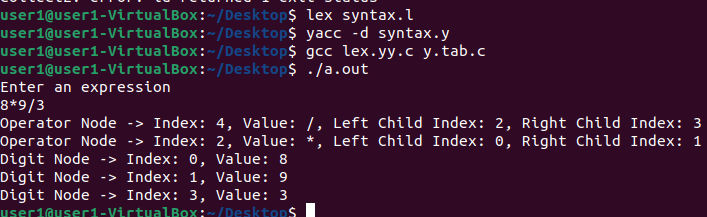
[\t] ;

[\n] return 0;

. return yytext[0];

%%

**Output – Screen shot**

****

Experiment No :06

**Aim of the program**

**Use YACC to generate 3-Address code for a given expression**

**Program**

**p.y**

%{

#include <math.h>

#include<ctype.h>

#include<stdio.h>

int var\_cnt=0;

char iden[20];

%}

%token digit

%token id

%%

S:id '=' E { printf("%s = t%d\n",iden, var\_cnt-1); }

E:E '+' T { $$=var\_cnt; var\_cnt++; printf("t%d = t%d + t%d;\n", $$, $1, $3 );

}

|E '-' T { $$=var\_cnt; var\_cnt++; printf("t%d = t%d - t%d;\n", $$, $1, $3 );

}

|T { $$=$1; }

;

T:T '\*' F { $$=var\_cnt; var\_cnt++; printf("t%d = t%d \* t%d;\n", $$, $1, $3 ); }

|T '/' F { $$=var\_cnt; var\_cnt++; printf("t%d = t%d / t%d;\n", $$, $1, $3 ); }

|F {$$=$1 ; }

;

F:P '^' F { $$=var\_cnt; var\_cnt++; printf("t%d = t%d ^ t%d;\n", $$, $1, $3 );}

| P { $$ = $1;}

;

P: '(' E ')' { $$=$2; }

|digit { $$=var\_cnt; var\_cnt++; printf("t%d = %d;\n",$$,$1); }

;

%%

int main()

{

var\_cnt=0;

printf("Enter an expression : \n");

yyparse();

return 0;

}

yyerror()

{

printf("NITW Error\n");

}

**p.l**

%{

#include<stdio.h>

#include<stdlib.h>

#include"y.tab.h"

extern int yylval;

extern char iden[20];

%}

d [0-9]+

a [a-zA-Z]+

%%

{d} { yylval=atoi(yytext); return digit; }

{a} { strcpy(iden,yytext); yylval=1; return id; }

[ \t] {;}

\n return 0;

. return yytext[0];

%%

**Output – Screen shot**

