VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

COURSE TITLE

Submitted by

AKSHANTH H M (1BM21CS014)

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

<u>CERTIFICATE</u>

This is to certify that the Lab work entitled "Compiler Design" carried out by **AKSHANTH HM (1BM21CS014),** 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.

Lohit JJ   **Dr. Jyothi S Nayak**

Assistant Professor   Professor and Head
Department of CSE   Department of CSE
BMSCE, Bengaluru   BMSCE, Bengaluru

`

Index

		D. 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

```python
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')
```

```python
            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

```
Enter your input! Enter blank next line to end: char a123 5 , +
char is a keyword!

a123 is an identifier!

5 is a number!

, is an invalid token!

+ is an operator!

Enter your input! Enter blank next line to end:
```
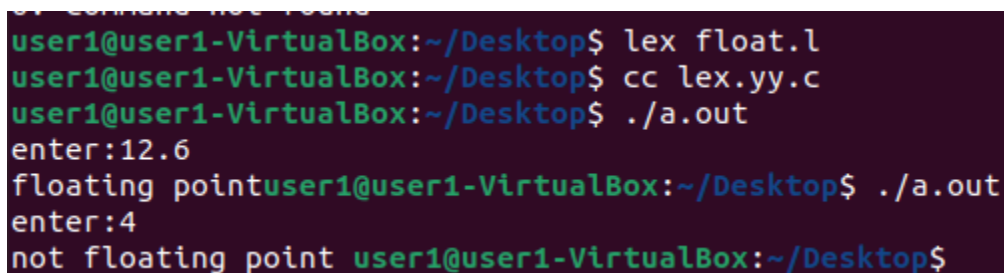
Experiment No :02

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


```
user1@user1-VirtualBox:~/Desktop$ lex float.l
user1@user1-VirtualBox:~/Desktop$ cc lex.yy.c
user1@user1-VirtualBox:~/Desktop$ ./a.out
enter:12.6
floating pointuser1@user1-VirtualBox:~/Desktop$ ./a.out
enter:4
not floating point user1@user1-VirtualBox:~/Desktop$
```

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(){}
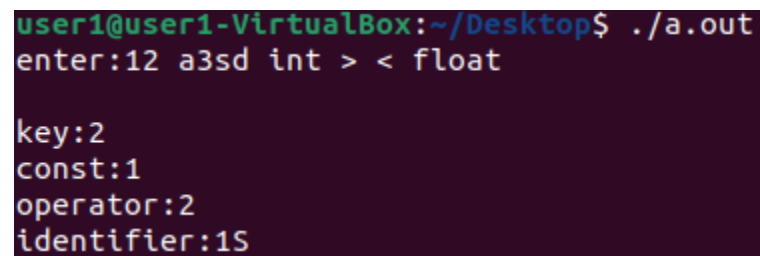
int main(){

printf("enter:");

yylex();

}

Output – Screen shot

```
user1@user1-VirtualBox:~/Desktop$ ./a.out
enter:12 a3sd int > < float

key:2
const:1
operator:2
identifier:1S
```

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

1 hi friend happy new year welcome to 2024 .

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)*(1
1|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

```
enter:^C
user1@user1-VirtualBox:~/Desktop$ lex p05.l
user1@user1-VirtualBox:~/Desktop$ cc lex.yy.c
user1@user1-VirtualBox:~/Desktop$ ./a.out
enter:100 122233 1000000001 1010 1234 2205

100 rule A
122233 rule B
1000000001 rule E
1010 rule D
1234 rule G
2205 rule F
```

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++;
```

```c
//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

```
1.S->cAd
2.A->ab/a
this is parser for the above grammar:
Enter any string:cdd
string is not accepted by the grammar
```

```
1.S->cAd
2.A->ab/a
this is parser for the above grammar:
Enter any string:cabd
string is accepted by the grammar
```

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;};
%%
```

```c
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;}
```
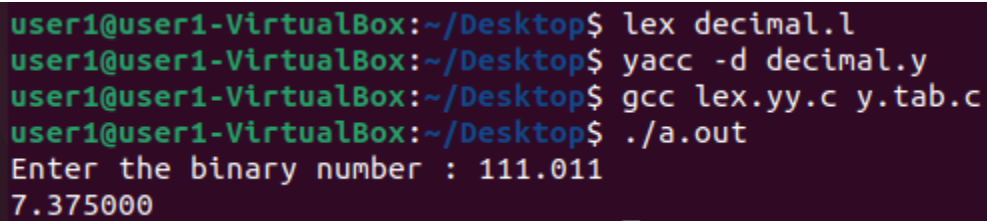
1 {yylval=1;return ONE;}

"." {return POINT;}

[\t] {;}

\n return 0;

%%

Output – Screen shot

```
user1@user1-VirtualBox:~/Desktop$ lex decimal.l
user1@user1-VirtualBox:~/Desktop$ yacc -d decimal.y
user1@user1-VirtualBox:~/Desktop$ gcc lex.yy.c y.tab.c
user1@user1-VirtualBox:~/Desktop$ ./a.out
Enter the binary number : 111.011
7.375000
```

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.I

```
%{

#include<stdio.h>

#include "y.tab.h"

extern int yylval;

%}


%%

[0-9]+ {

        yylval=atoi(yytext);

        return NUMBER;

        }

[\t] ;
```
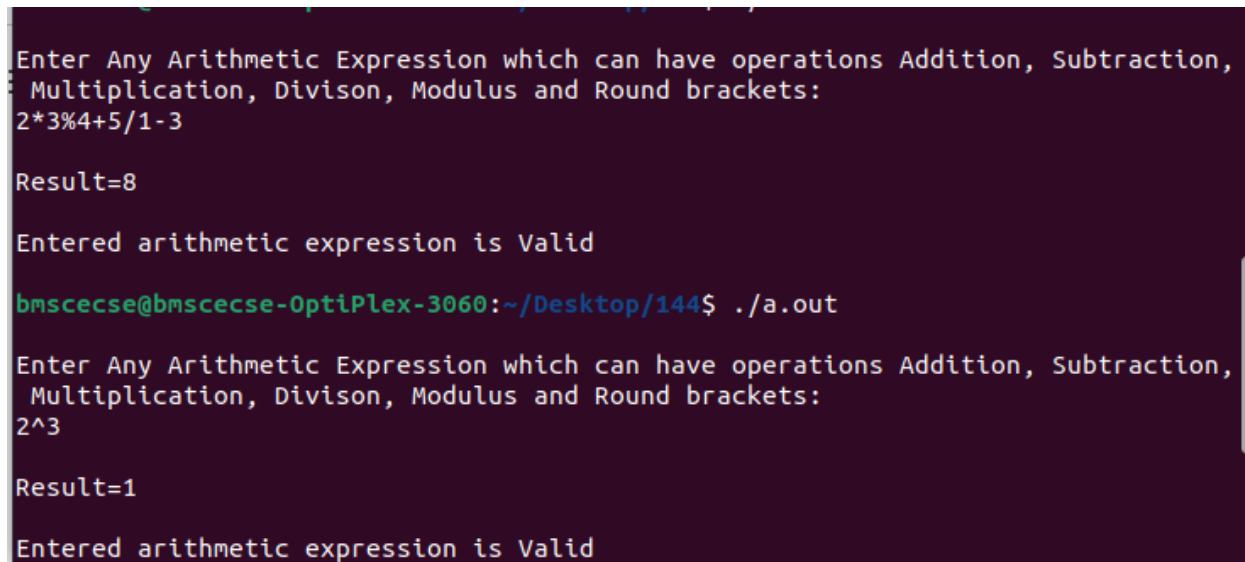
[\n] return 0;

. 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"
```
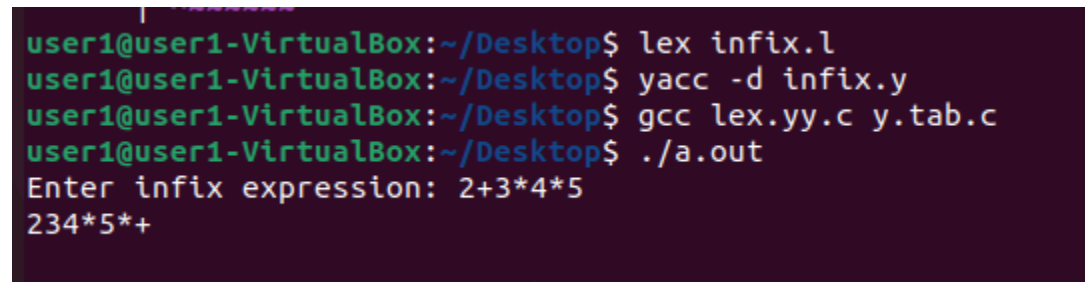
extern int yylval;

% }

% %

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

[\t] ;

[\n] return 0;

. return yytext[0];

% %

Output – Screen shot

```
user1@user1-VirtualBox:~/Desktop$ lex infix.l
user1@user1-VirtualBox:~/Desktop$ yacc -d infix.y
user1@user1-VirtualBox:~/Desktop$ gcc lex.yy.c y.tab.c
user1@user1-VirtualBox:~/Desktop$ ./a.out
Enter infix expression: 2+3*4*5
234*5*+
```

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

```
user1@user1-VirtualBox:~/Desktop$ lex syntax.l
user1@user1-VirtualBox:~/Desktop$ yacc -d syntax.y
user1@user1-VirtualBox:~/Desktop$ gcc lex.yy.c y.tab.c
user1@user1-VirtualBox:~/Desktop$ ./a.out
Enter an expression
8*9/3
Operator Node -> Index: 4, Value: /, Left Child Index: 2, Right Child Index: 3
Operator Node -> Index: 2, Value: *, Left Child Index: 0, Right Child Index: 1
Digit Node -> Index: 0, Value: 8
Digit Node -> Index: 1, Value: 9
Digit Node -> Index: 3, Value: 3
user1@user1-VirtualBox:~/Desktop$
```

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

```
user1@user1-VirtualBox:~/Desktop$ lex code3.l
user1@user1-VirtualBox:~/Desktop$ yacc -d code3.y
user1@user1-VirtualBox:~/Desktop$ gcc lex.yy.c y.tab.c
user1@user1-VirtualBox:~/Desktop$ ./a.out
Enter an expression :
result=2+3*4
t0 = 2;
t1 = 3;
t2 = 4;
t3 = t1 * t2;
t4 = t0 + t3;
result = t4
```