**11. Lab Manual**

**Tutorial – 1:- implement lexical analyzer in c.**

#include<stdio.h>

#include<string.h>

#include<ctype.h>

#include<conio.h>

void main()

{

int i,a[100],len,count,l;

char s[100],j[10],s1[100],var[100];

clrscr();

printf("Enter the string:");

gets(s);

len=strlen(s);

for(i=0;i<len;i++)

{

if(isalpha(s[i]))

{

if(s[i]=='i' && s[i+1]=='n' && s[i+2]=='t')

{

printf("%c%c%c is keyword \n ",s[i],s[i+1],s[i+2]);

i=i+2;

}

else if(s[i]=='f' && s[i+1]=='o' && s[i+2]=='r')

{

printf("%c%c%c is keyword \n ",s[i],s[i+1],s[i+2]);

i=i+2;

}

else if(s[i]=='w' && s[i+1]=='h' && s[i+2]=='i' && s[i+3]=='l')

{

printf("%c%c%c%c is keyword \n ",s[i],s[i+1],s[i+2],s[i+3]);

i=i+3;

}

else if(s[i]=='a' && s[i+1]=='u' && s[i+2]=='t' && s[i+3]=='o')

{

printf("%c%c%c%c is keyword \n ",s[i],s[i+1],s[i+2],s[i+3]);

i=i+3;

}

else if(s[i]=='c' && s[i+1]=='a' && s[i+2]=='s' && s[i+3]=='e')

{

printf("%c%c%c%c is keyword \n ",s[i],s[i+1],s[i+2],s[i+3]);

i=i+3;

}

else if(s[i]=='c' && s[i+1]=='h' && s[i+2]=='a' && s[i+3]=='r')

{

printf("%c%c%c%c is keyword \n ",s[i],s[i+1],s[i+2],s[i+3]);

i=i+3;

}

else if(s[i]=='e' && s[i+1]=='l' && s[i+2]=='s' && s[i+3]=='e')

{

printf("%c%c%c%c is keyword \n ",s[i],s[i+1],s[i+2],s[i+3]);

i=i+3;

}

else if(s[i]=='b' && s[i+1]=='r' && s[i+2]=='e' && s[i+3]=='a' && s[i+4])

{

printf("%c%c%c%c is keyword \n ",s[i],s[i+1],s[i+2],s[i+3],s[i+4]);

i=i+4;

}

else

printf("%c is identifier \n ",s[i]);

}

else if(isdigit(s[i]))

{

printf("%c",s[i]);

while(isdigit(s[i+1]) || s[i+1]=='.')

{

printf("%c",s[i+1]);

i++;

}

printf(" is digit \n");

}

else if(s[i] == '+' || s[i] == '-' || s[i] == '\*' || s[i] == '/')

{

if(s[i+1] == '+')

{

printf("%c%c is increment operator \n",s[i],s[i+1]);

i++;

}

else if(s[i+1] == '-')

{

printf("%c%c is decrement operator \n",s[i],s[i+1]);

i++;

}

else

{

printf("%c is arithmetic operator \n ",s[i]);

}

}

else if(s[i] == '=')

{

if(s[i+1] == '=')

{

printf("%c%c is relational operator \n",s[i],s[i+1]);

i++;

}

else if(s[i-1] == '!')

{

printf("%c%c is relational operator\n ",s[i],s[i-1]);

i++;

}

else

{

printf("%c is assignment operator \n ",s[i]);

}

}

else if(s[i] == '<' || s[i] == '>')

{

if(s[i]=='>' && s[i+1] == '>' || s[i] == '<' && s[i+1] == '<')

{

printf("%c%c is bitwise operator \n", s[i],s[i+1]);

i++;

}

else if(s[i+1] == '=')

{

printf("%c%c is relational operator\n ",s[i],s[i+1]);

i++;

}

else

{

printf("%c is relational operator \n",s[i]);

}

}

else if(s[i] == '&' || s[i] == '|' || s[i] == '^' || s[i] == '~')

{

printf("%c is bitwise operator \n", s[i]);

}

}

getch();

}

**Output:-**

****

**Tutorial – 2:- implement regular expression in c.**

**1. a\***

#include<stdio.h>

#include<conio.h>

#include<string.h> void main()

{

char s[10],flag; int i=0; clrscr();

printf("Enter The String For a\*:"); gets(s);

while(s[i]!='\0')

{

if(s[0]=='\0')

{

flag=1;

}

if(s[i]=='a')

{

flag=1;

}

else

{

flag=0;

break;

}

i++;

}

if(flag==1)

{

printf("Valid");

}

else

printf("In Valid");

getch();

}

Output –



**2. a\*b**

#include<stdio.h>

#include<conio.h>

#include<string.h> void main()

{

int flag,length,i=0; char str[15]; clrscr();

printf("\nEnter The String For a\*b:\n"); gets(str);

if(strlen(str)==1)

{

if(str[0]=='b')

{

flag=0;

}

else

{

flag=1;

}

}

if(strlen(str)>1)

{

length = strlen(str); while(str[i]!='\0')

{

if(str[length-1]=='b'&& str[i]=='a')//&& i==strlen(str)-2)

{

flag=0;

}

else

{

flag=1;

break;

}

i++;

//if()

}

}

if(flag==0)

{

printf("Valid \n"); }

if(flag==1)

{

printf("Invalid \n");

}

getch();

}

Output –



**3. (a/b)\***

#include<stdio.h>

#include<conio.h>

#include<string.h>

void main()

{

char str[8],is; int i=0;

char state='x';

clrscr();

printf("\n Enter String :"); gets(str);

for(i=0;i<strlen(str);i++)

{

is=str[i];

if(state=='x' && is=='a') state='x';

else if(state=='x' && is=='b') state='x';

}

if(state=='x')

printf("\n Valid String");

else

printf("\n Invalid String");

getch();

}

Output –



**4. 00(0/1)\***

#include<conio.h>

#include<string.h>

void main()

{

char str[8],is; int i=0;

char state='x';

clrscr();

printf("\n Enter String :"); gets(str);

for(i=0;i<strlen(str);i++)

{

is=str[i];

if(state=='x' && is==0) state='y';

else if(state=='x' && is==1) state='f';

else if(state=='y'&& is==0) state='z';

else if(state=='y' && is==1) state='f';

else if(state=='z' && is==0) state='z';

else if(state=='z' && is==1) state='z';

}

if(state=='z')

{

printf("\nValid String");

}

else

{

printf("\nInvalid String");

}

getch();

}

Output -



**5. (a/b)\*abb**

#include<stdio.h>

#include<conio.h>

#include<string.h> void main()

{

char str[8],is; int i=0,length=0; char state='x';

clrscr();

printf("\n Enter String :"); gets(str);

for(i=0;i<strlen(str);i++)

{

is=str[i];

if(state=='x' && is=='a') state='x';

else if(state=='x' && is=='b') state='x';

}

length = (strlen(str));

if(str[length-3]=='a' && str[length-2]=='b' && str[length-1]=='b')

{

state='x';

}

else

{

state='f';

}

if(state=='x')

printf("\n Valid String");

if(state=='f')

printf("\n Invalid tring");

getch();

}

Output -



**6. (0/1)\*01(0/1)\***

#include<stdio.h>

#include<conio.h>

#include<string.h>

void main()

{

char str[8],is; int i=0;

char state='x';

clrscr();

printf("\n Enter String :"); gets(str);

for(i=0;i<strlen(str);i++)

{

is=str[i];

if(state=='x' && is=='0') state='y';

else if(state=='x' && is=='1') state='x';

else if(state=='y'&& is=='0') state='y';

else if(state=='y' && is=='1') state='z';

else if(state=='z' && is=='0') state='z';

else if(state=='z' && is=='1') state='z';

}

if(state=='z')

{

printf("\nValid String");

}

else

{

printf("\nInvalid String");

}

getch();

}

Output -



**7. anbn ,n>1**

#include<stdio.h>

#include<conio.h>

#include<string.h> void main()

{

char str[8],a,b;

int i=0,counter=0,flag=0;

clrscr();

printf("\nEnter The String:"); gets(str);

while(str[i]=='a')

{

counter++;

i++;

}

while(str[i]=='b')

{

counter--; i++;

}

if(str[i]=='a'|| str[i]=='b')

{

flag=1;

}

if(counter==0 && flag==0)

{

printf("\nValid Sring.");

}

if(flag==1)

{

printf("\nInvalid String.");

}

getch();

}

Output -



**8. anbm ,n>=m**

#include<stdio.h>

#include<conio.h>

#include<string.h>

void main()

{

char str[8],a,b;

int i=0,counter=0,flag=0; clrscr();

printf("\nEnter The String:"); gets(str);

while(str[i]=='a')

{

counter++;

i++;

}

while(str[i]=='b')

{

counter--; i++;

}

if(str[i]=='a'|| str[i]=='b')

{

flag=1;

}

if(counter>=0 && flag==0)

{

printf("\nValid Sring.");

}

if(counter<0 || flag==1)

{

printf("\nInvalid String.");

}

getch();

}

Output –



**9. anb2m ,n=m**

#include<stdio.h>

#include<conio.h>

#include<string.h>

void main()

{

char str[8],a,b;

int i=0,counter1=0,counter2=0,flag=0; clrscr();

printf("\nEnter The String:"); gets(str);

while(str[i]=='a')

{

// counter++; //2

counter1++; //2 i++;

}

while(str[i]=='b')

{

// counter--; //-2 counter2++; //4 i++;

}

if(str[i]=='a'|| str[i]=='b')

{

flag=1;

}

if((counter2==2\*counter1) && flag==0)

{

printf("\nValid Sring.");

}

if((counter2!=2\*counter1) || flag==1)

{

printf( "\nInvalid String.");

}

getch();

}

Output –



**10. anbm , n is odd and m is even**

#include<stdio.h>

#include<conio.h>

#include<string.h> void main()

{

char str[8],a,b;

int i=0,counter1=0,counter2=0,flag=0; clrscr();

printf("\nEnter The String:"); gets(str);

while(str[i]=='a')

{

counter++; //2 counter1++; //2 i++;

}

while(str[i]=='b')

{

// counter--; //-2 counter2++; //4 i++;

}

if(str[i]=='a'|| str[i]=='b')

{

flag=1;

}

if(counter1%2!=0 && counter2%2==0 && flag==0)

{

printf("\nValid Sring.");

}

else

{

printf( "\nInvalid String.");

}

getch();

}

Output –



**Tutorial – 3:- implement Finite Automata in c.**

//Finite automata for a\*b

#include<stdio.h>

#include<conio.h>

void main()

{

int state[2],i,j,c,d; char isymbol[4]; clrscr();

printf("Enter No. of State for a\*b:");

for(i=0;i<2;i++)

{

scanf("%d",&state[i]);

}

printf("Enter No. of I/p Symbol for a\*b:"); for(j=0;j<2;j++)

{

scanf("%s",&isymbol[j]);

}

//finite automata for a\*b

printf("\ta \t b\n"); printf(" --------------"); printf("\ns1\n\ns2");

for(c=0;c<2;c++)

{

for(d=0;d<2;d++)

{

if(state[c]==1 && isymbol[d]=='a')

{

gotoxy(8,7);

printf("%d",state[c]);

}

if(state[c]==1 && isymbol[d]=='b')

{

gotoxy(18,7);

printf("%d",state[d]);

}

if(state[c]==2 && isymbol[d]=='a')

{

gotoxy(8,9);

printf("%d",state[d]);

}

if(state[c]==2 && isymbol[d]=='b')

{

gotoxy(18,9);

printf("%d",state[0]);

}

}

}

getch();

}

Output –



**Tutorial – 4:- implement lexical analyzer using LEX tool.**

%{

#include<stdio.h> int main(void)

{

yylex(); return 0;

}

%}

%option noyywrap

%%

[0-9]+ printf("\n%s\tInteger",yytext);

"if"|"else"|"int"|"char"|"scanf"|"printf"|"switch"|"return"|"struct"|"do"|"while

"|"void"|"for"|"float" printf("\n%s\t is keyword",yytext);

[A-Za-z][\_]\*[A-Za-z0-9]+|[A-Za-z]d printf("\n%s\tVariable",yytext); [0-9]+"."[0-9]+ printf("\n%s\t is floating pt no ",yytext);

"&&"|"<"|">"|"<="|">="|"="|"+"|"-"|"?"|"\*"|"/"|"%"|"&"|"||" printf("\n%s\toperator ",yytext); "{"|"}"|"["|"]"|"("|")"|"#"|"'"|"."|"\""|"\\"|";"|"," printf("\n%s\t is a special character",yytext);

%%

Output –

int a[10];

int is keyword a is variable

[ is a special character

10 is integer

] is a special character

; is a special character

**Tutorial – 5:- implement syntax analyzer using YACC tool.**

**Cacl.y**

%{

#include <stdio.h>

int regs[26];

int base;

%}

%start list

%token DIGIT LETTER

%left '|'

%left '&'

%left '+' '-'

%left '\*' '/' '%'

%left UMINUS /\*supplies precedence for unary minus \*/

%% /\* beginning of rules section \*/

list: /\*empty \*/

|

list stat '\n'

|

list error '\n'

{

yyerrok;

}

;

stat: expr

{

printf("%d\n",$1);

}

|

LETTER '=' expr

{

regs[$1] = $3;

};

expr: '(' expr ')'

{

$$ = $2;

}

|

expr '\*' expr

{

$$ = $1 \* $3;

}

|

expr '/' expr

{

$$ = $1 / $3;

}

|

expr '%' expr

{

$$ = $1 % $3;

}

|

expr '+' expr

{

$$ = $1 + $3;

}

|

expr '-' expr

{

$$ = $1 - $3;

}

|

expr '&' expr

{

$$ = $1 & $3;

}

|

expr '|' expr

{

$$ = $1 | $3;

}

|

'-' expr %prec UMINUS

{

$$ = -$2;

}

|

LETTER

{

$$ = regs[$1];

}

|

number

;

number: DIGIT

{

$$ = $1;

base = ($1==0) ? 8 : 10;

} |

number DIGIT

{

$$ = base \* $1 + $2;

}

;

%%

main()

{

return(yyparse());

}

yyerror(s)

char \*s;

{

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

}

yywrap()

{

return(1);

}

**Calc.l**

%{

#include <stdio.h>

#include "y.tab.h"

int c;

extern int yylval;

%}

%%

" " ;

[a-z] {

c = yytext[0];

yylval = c - 'a';

return(LETTER);

}

[0-9] {

c = yytext[0];

yylval = c - '0';

return(DIGIT);

}

[^a-z0-9\b] {

c = yytext[0];

return(c);

}

Output –

Example-1

5 + 3

8

Example-2

6 \*

Syntax invalid