Experiment-1 Sindhu Vardhan

Exp-1: Develop a lexical Analyzer to identify identifiers constants, operators using C program

Aim: To develop lexical Analyzer to identify Constant operator using C program.

Algorithm:
- Start the code and import the package
- Declare the variable
- Assign the variable
- Save and Run the code.

Program:

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
int main()
{
int i, c=0, m, cc=0, oc=0, j;
Char b[30], operator[30], identifier[30], Constant[30]
Printf("Enter the string.");
Scanf("%s", &b);
for(i=0; i<strlen(b); i++)
{
  if(isspace(b[i]))
  {
    Continue;
  }
  else if(isalpha(b[i]))
  {
    identifier[ic] = b[i]
```

```
        i++
    }
    else if (isdigit(b[i]))
    {
        m=(b[i]-'0');
        i=i+1
        while (isdigit(b[i]))
        {
            m=m*10+(b[i]-'0');
            i++;
        }
        i=i-1;
        Constant[cc]=m;
            c++;
    }
    else
        if(b[i]=='*')
        {
            operator[oc]='*';
            oc++;
        }
        else if(b[i]=='-')
        {
            operator[oc]='+';
            oc++;
        }
        printf("identifier:");
        for(j=0;j<i;j++)
        {
            printf("%c",identifier[j]);
        }
        printf("\n Constant");
        for(j=0;j<cc;j++)
        {
            printf("%d",Constant[j]);
        }
```

```
Printf("\n operator ");
for(j=0;j<coc;j++)
{ printf("%c", operator[j]);
}
}
```

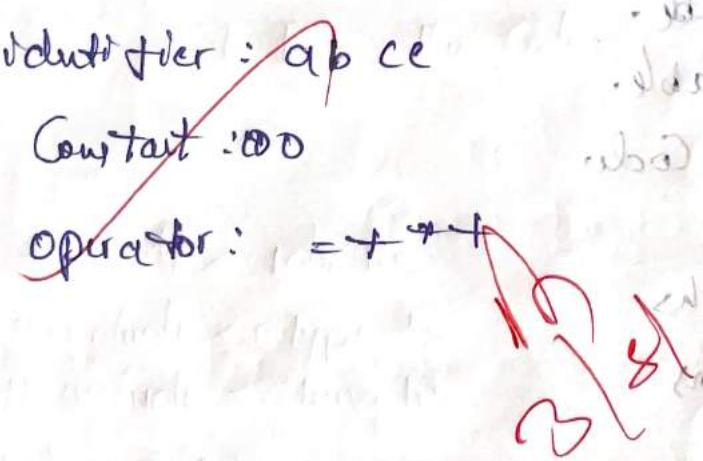
Output:

Enter string: a=b+c*c+100

Identifier : a b c e

Constant : 100

operator : = + * +



Exp-2: Develop a lexical Analyser to identify weather a given line Comment or not using C.

Aim: To develop a lexical Analyser to identify weather comment or not using C.

Algorithm:

Start the Code and import package.
Declare the variable.
Assign the variable.
Save & Run the Code.

Program:

```
#include <stdio.h>
#include <conio.h>
int main()
{
 Char Com[50]; i=2,a=0;
 printf("\n Comment:");
 gets(com);
 if(Com[0]== '/')
 {
   if(Com[1]=='/')
     printf("\n it is Comment");
   else if(Comm[1]==*')
   {for(i=2; i<50;i++)
    {
      printf("\n is Comment");
      a=1;
```

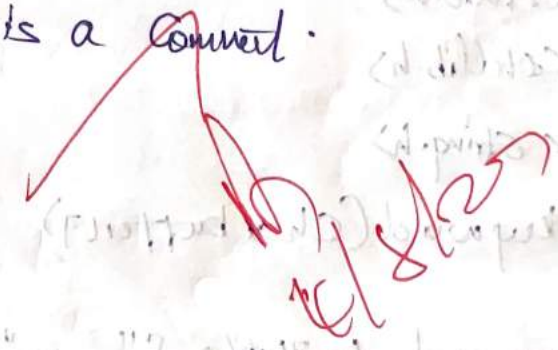
```
        break;
    }
}
```

Output :

Input : Enter :

Comment : // hello

Output : It is a comment.



Experiment 3 — Sindhu Vardhan

③ Design a lexical Analyzer for given language should ignore the redundant space, tab and new and ignore Comment.

Aim: To develop lexical Analyzer for given language should ignore the redundant space, tab, Comment using C.

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include < string.h>
int is keyword (Char buffer[])
{
  Char keyword [32][10] = {"main", "auto", "break",
    "Case", "char", "Const", "Continue", "default", "do"
    double", "else" int};
    int i, flag = 0
  for (i=0 ; i<32 ; ++i)
  {
    if (strcmp (keyword [i], buffer) == 0){
      flag = 1;
      break;
    }
  }
  return flag;
}
```

```
int main()
{
char ch, buffer[15], operators[] = "+-*/%=";
FILE *fp;
int i,j=0;
if ((fp=fopen("program.txt","r"))==NULL)
{
printf("error");
}
while ((ch=fgetc(fp))!=EOF)
{
if (ch==operators[i])
printf("%c is operator\n",ch);
}
```

Input: program.txt

```
int main()
{
int a,b,c;
printf("Hello");
}
```

Output:

int is keyword
a is identifier
b is identifier
c is identifier.

Experiment-4 Sindhu Vardhan

Aim: To develop lexical Analyser to validate operator to recognize operation +, -, *, /, using C.

Program:

```
#include <stdio.h>
#include <conio.h>
int main()
{ Char s[5]
  Printf("\n enter any operator)
  get(s);
  Swith(s[0])
  {
    Case '>': if(s[1]=='=')
                printf("Greater than or equal")
              else
                print("\n greater")
    Case '<': if(s[1]=='=')
                Print("\n less than or equal")
              else
                printf("less than");
                break;
    Case '=': if(s[1]=='=')
                Printf("equal");
              else
                printf("not equal");
```

```
Case '&': if (str == '&')
              printf("AND")

        else printf ("Bitwise AND")

        break;

Case '+':  printf ("Addition");
           break;
Cas '-' :  printf ("subraction"

 default :  printf("Invalid");

}
}
```

<u>Output</u>

Enter the the operator: <=

=> Less than or equal to

Experiment-05

Sindhu Vadla

Aim: To design lexical Analyzer to find no of white space and newline Character.

Program:

```
#include <stdio.h>
int main()
{
  Char str[100];
  int word=0, newline=0, Char=0;
  Scanf("%. Lan]", &str];
  for(int i=0; str[i]!='\0'; i++)
  {  if(str[i]==" ")
     {
       word++;
     }
     else if (str[i] == '\n')
     {
        newline++;
        word++;
     }
     else if (str[i]!=" && str[i]!='\n')
     {
       Char++
     }
  }
if (char>0)
{
  word++;
  newline++;
```

```
    printf(" Total word : %d ", word);
    printf (" Total lines %d", lines);
    printf (" Total Character %d", Char);
    return 0;
}
```

Output:

```
void main()
{
  int a,b;
  a=b+c;
  c= d*e;
}
```

Total words: 11

Total line: 7

Experiment-6

Aim: To develop lexical analyser to test given identifier is valid or not using C.

Program:

```
#include <stdio.h>
#include <ctype.h>
#include <conio.h>
int main()
{
  char a[10];
  int flag, i=1;
  printf("Enter identifier:");
  gets(a);
  if (isalpha(a[0]))
  {
     flag=1;
  }
  else
     printf("Not a valid identifier");
     while (a[i]!='\0')
  {
  if (flag==0)
  { break;
  }i++;
  }
}
```

Output:

Enter identifier : abc123

valid identifier:

Experiment 7 — Sindhu Vardhan

Aim: To write a C program to find FIRST() - Predictive parser for given grammar.

Program:

```
#include<stdio.h>
#include<ctype.h>
void first(char[],char);
void addToResultSet(char[],char);
int numOfProduct;
char productionSet[10][10];
int main()
{
  int i;
  char choice;
  char c;
  char result[20];
  printf("How no of production");
  scanf("%d", &noOfProduction);
  for(i=0;i<noOfProduction;i++)
  {
    printf("Enter no of production No %d:",i+1);
    scanf("%s",productionSet[i]);
  }
  do
  {
    printf("\n find first of:");
    scanf("%c",&c);
    first(result,c);
```

```
first(out,c);
printf("first(%c)= {",c);
for(i=0; result[i]!='\0'; i++)
printf("%c", result[i]);
printf("}\n");
}
void first(char *result,char c)
{
int i,j,k;
char subresult[20];
int found posion;
subresult[0]='\0';
{
add to result set(result,c);
return;
break;
}i++
}
```

Output:

How many no of production=2

Enter production 1: S=AaAb

Enter production 2: S=BbBa

Find the first:-

first(S)= {$ab}

press y to continue

Aim: To find follow() predictive parser for given grammer using C.

Program:

```
#include <stdio.h>
#include <string.h>
int limit, x=0;
Char production [10][10], array[10];
void find_first (Char ch)
void find_follow (Char ch);
void Array_manipulation (Char ch);
int main()
{
  int count;
  Char option, ch;
  Printf ("production:");
  Scanf ("%d", &limit);
  for (count=0; count<limit; count++)
  {printf ("Enter [%d]", count);
  Scanf ("%s", production [count]);
}
do
{ x=0;
  Printf ("Enter value of find:");
  Scanf ("%c", &ch);
  Printf ("follow value of %c:");
```

```
Printf("%s\n",%
Printf("\n to continue");
Scanf("%c", &option);
}while (option == 'y' || option == 'Y')
  return 0;
}
void find_follow (char ch)
{ int i,j;
  int length = strlen (production[i]);
  if (production[0][0]==ch)
  {
    array_manipulation ('$');
  }
  for(i=0; i<limit; i++)
  {
    for(j=2; j<length; j++)
    { if (arr[i][count]==ch)
      { return;
      }
    }
  }
}
```

Output

Enter production:2

value of production[1] = S = AaAb

value of production[2] = S = BbBa

Experiment - 9 Siddhartha

Aim: To implement c program to eliminate left recursion from given CFG;

Algorithm:

```
#include <stdio.h>
#include <stdlib.h>
# define size 10
int main ()
{
  Char non_terminal;
  Char b, a;
  int n;
  Char P [10] [5];
  int indx = 3;
  printf("Enter no of production:");
  scanf("%d", &num);
  Printf("enter grammer, as E->E-A");
  for (int i=0; i<num; i++)
  {
    scanf("%s", P[i]);
  }
  for (int i=0; i<n; i++)
  {
    Printf("Grammer: "production [i]);
    non_terminal = production [i] [0];
  if (P[i] [indx] != 0)
  {
    printf(" non_terminal -> alpha. non_terminal);
  }
```

```
else
    printf("Can't reduced:");
}
else
    printf("%s is not left recursive",
    index);
}
}
```

output:

Enter no of productions:2

Enter the grammer as E→E+A;

S→(L)|a

L→L,S|S

GRAMMER::::S→(L)|a is not left recursion

GRAMMER::L→L,S|S is left recursive.

Aim: To implement a C program to eliminate left factoring from given CFG.

Program:

```
#include<stdio.h>
#include<string.h>
int main()
{
  Char gram[20], part1[20], part2[20], mg[20],
    ng[20], tg[20];
  int i,j=0, k=0, l=0, pos;
  Printf("Enter production:");
  get(gram);
  for(i=0; gram[i]!='|'; i++, j++)
  {
      part1[j] = gram[i];
      part1[j] = '\0';
  for(j=++i; i=0; gram[i]!='\0';
                  j++; i++).
  {
    Part[i] = g[j];
  for(i=0; i<strlen(part1)||i<strlen(part2)
                                  i++)
  { if(p1[i]==p2[i])
  {
```

```
Case 2 : display();
         break;

Case 3 : printf ("Enter label");
         scanf ("%s", l);
         val = search(l);
         if (val == l)
         { printf ("label found")
         }

Case 4 : Modify();
         break;

Case 5 : Exit(0);
         break;

}
}
```

Output:

Enter a label : a

Enter the address : 100

=) 5

Exit.

Experiment-12 Siddhartha

Aim: To write a C program to, Construct the recursive descent for grammer.

Program:

```
#include <stdio.h>
#include <stdlib.h>
Char in[100]
int i, l;
void main()
{
  printf("recursive parsing:");
  printf("E → TE' | E' → +TE' @ T → FT");
  printf("Enter string:");
  gets(input);
  if(E())
  {
    if (input(i+1)=='\0')
    printf("string accepted");
  else
    printf("not accepted");
```

Output:

Recursive descent parsing

E → TE', E' → +TE'

@ String not accepted.

Experiment-13 Sindhuja Vardhan

Aim: write c program to check grammer or not for Top-Down parsing or Bottom-up parsing.

Program:

```
#include<stdio.h>
#include<string.h>
void main()
{
  Char string[50];
  int flag, count=0;
  Printf("The grammer is → S→aA, S→Sb, S→ab");
  Printf("Enter string:");
  gets(string);
  if(string[0]=='a')
  if
    flag=0;
    for(count=1; string[count]!=0; count++)
    {
    if(string[count]=='b')
    {
      flag=1;
      Continue;
    }
    printf("The string:");
      break;
    else
      Printf("String accepted:");
}
```

Output:

The grammer is: S→aA, S→Sb→S→ab.

enter string: abb

String accepted.

Experiment-14 Siddhartha

Aim: To implement the shift reduce parsing using C program.

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
Char ip_sym[15], stack[15]; int ip_ptr=0, st_ptr
Char temp[2], act[15];
void check(); int main()
{
  Printf("shift reduce parse");
  Printf("Grammer");
  Printf("E→E+E"); printf(E→E*E E→a/b)
  Printf("Enter the input symbol");
  gets(ip_sym);
  Printf("stack table");
  Printf("stack");
temp[0]= ip_sym[ip_ptr]; temp[1]="\0"
strcat(act, temp);
{
  stack[st_ptr] = ip_sym[ip_ptr];
  stack[str_ptr] = '\0'; ip_sym[ip_ptr]=
  Printf("\n\t%s\t%s\t", act
```

```
printf(ip,sup,act)
temp[0] = ip->str[ip->ptr];
st->ptr++;
}
st->ptr++;
}
st->ptr++; Check();
}
void Check;
if(flag==0)
{
printf("reject", stack, ID_syn);
exit(0)
}
}
```

Output:

Shift Reduce Parse

Grammar

$E \rightarrow E+F$

$E \rightarrow E/E$

$E \rightarrow E*F$

$E \rightarrow a/b$

Input = a+b

Experiment-15

Aim: To implement the operator precedence parsing using C program.

Program:

```
#include <stdio.h>
#include <string.h>
Char * input;
int i=0;
Char lasthandle[6], stack[50], handle[][5]
  = {"E+E", "E-E", "E*E", "E/E", "E^E"};
int top=0,
int Char prec[9][9]= {
int get index (char c)
{
  switch(c)
  {
    Case '+': return 0;
    Case '-': return 1;
    Case '*':
        return 2;
    Case '/':
        return 3;
    Case '^':
        return 4;
    Case (i): return 5;
    Case 'C': return 6;
    Case ')': return 7;
    Case '$': return 8;
```

```
int shift()
{
  stack[++top] = *(input + i++);
  stack[top+1] = '\0';
}
int reduce()
{
  int i, len, found, t;
  for(i=0; i<5; i++)
  {
    len = strlen(handle[i]);
    if(stack[top] == handle[i][0] && top+1 == len)
    {
      found = 0;
      break;
    }
  }
  if(strcmp(stack, "$E$") == 0)
    printf("Accepted");
  else printf("not accepted");
}
```

Output

Stack	Input	Action
$i	*(i+i)i$	Shift i
$E	*(i+i)i$	Red E→i
$E*	(i+i)i$	Shift
$E*(	i+i)i$	Shift
$E*(i	+i)i$	Shift

Experiment-6

Aim: To generate three address code representation for given input string.

Program:

```
#include <stdio.h>
struct three
{
  char data[10], temp[7];
} s[20];
int main()
{
  char d[7], d2[7] = "t";
  int i=0, j=1, len=0
  f1 = fopen("sum.txt", "r")
  f2 = fopen("out.txt", "w")
  while (fscanf(f1, "%s", s[len].data) != EOF)
    len++;
  i=0; (j, d, i++);
if (!strcmp(s[3].data, "+"))
{
  fprintf(f2, "%s = %s + %s", s[j].temp, s[j+2].data);
  j++;

{
else if (!strcmp(s[3].data, "-"))
{
  fprintf(f2, "%s = %s - %s", s[j].temp);
  j++;
```

```
for(i=0;i<len;i+=2)
{
  itoa(j,d1,t)
  strcat(d2,d1);
  strcat(s[j].temp,d2);
  if(!strcmp(s[i+1].data)
  fprint(f2,"%s=%s");
  else
    strcpy(d1,'t');
    strcpy(d2,"t");
     j+1
  }
  getch();
}
```

Output:

Input: int1 + int2 - int3

t_1 = int1 + int2

t_2 = t_1 - int3

out = t_2;

Experiment - 17

Aim: To implement the lexical analyzer to scan & count no of Character, word using C.

program:

```
#include <stdio.h>
int main()
{
Char Str[100];
int word=0, newline=0, Character=0;
scanf ("%.L[n]", &Str);
for(int i=0; Str,Li] != '\0', i++)
{
  if (Str Li] == ' ')
    {
      word++;
    }
    else if (Str Li] == '\n')
    {
      newline++;
      word ++;
  }
Printf ("Total word", word);
Printf ("Total line %d", line);
Printf ("Total Character %d", Chater);
}
```

Output:

```
void main()
{ int a, b, c;
    c=a+b;
}
Total no wor=4
}
```

Experiment-18

Aim: To implement the back end of Compiler using c program.

Program:

```
#include<stdio.h>
#include<string.h>
void main()
{ int n,i,j;
 Char a[50][50];
 Printf("Enter no: intermediate:");
 Scanf("%d",&n);
 for(i=0;i<n;i++)
 {
  Printf("Enter Code a %d:",i+1)
  for(j=0;j<6;j++)
  {
   Scanf("%c",&a[i][j]);
  }
}
Print("The generated is:");
for(i=0;i<n;i++)
{
Print("mov r.c,%r.d",a[i][5]);
```

```
if (a[i][4]== '-')
{
 printf("Sub %c,R%d", a[i][5], i);
}
if (a[i][4]== '+')
{
 printf("Add %c,R%d", a[i][5], i);
}
return 0;
}
```

Output:

Enter no intermediate code: 2

Enter 3 address 1 : a=b+c.

Enter 3 address 2 : d=m+d

generated Code.

```
MOV b,R0
add c,R0
MOV, R0,a
MOV. m,R1
MOV d,R1
MOV R1,d.
```

Experiment-19

Aim: To write C program to Computer LE
operator precedence parser for grammer.

Program:

```
#include <stdio.h>
#include <conio.h>
Char arr[18][3]={{'E','+','F'},{'E','*','F'},
{'E','C','F'},{'E'},{'E'},{'F'},{'F'},
Char prod[]="EETTFF";
Char ire[6][3]={{"E","+","T"},{"T"},
{"T","*","F"},{"F"},{"\0"}};
Char stack[5][2]
int top=-1;
void install (Char pro, Char re)
{ int i;
for (i=0;i<18;i++)
 {
  if (arr[i][0]==pro && arr[i][i]==re
  {
   arr[i][0]='T';
    break;
 }
}
```

```
    ++top;
    stack[top][0]=p[0];
    stack[top][1]=r[i];
}
int main()
{
    int i=0,j;
    char p[0],r[i],pi='i';
    for(j=0;j<6;j++)
    {
        for(j=0;j<3&&(r[i][j]!='\0');++j)
        {
            if(r[i][j]=='E'||r[i][j]=='j')
                break;
        }
    }
}
```

Output:

E++

E*T

E(T

E)F

Ei+

E$f

F$f

f+f

E → +TE

E → TE

T → *FT

Experiment-20

Aim: To write c program to compute TRAIL operator parser for grammer.

program:

```
#include <stdio.h>
#include <conio.h>
Char arr[][3] = {{'E', '+', 'T'}, {'E', '*', 'T'},
   {'E', 'C', 'T'}, {'E', ')', '+'}
Char prod[8] = "FETTFF"
Char re[6][3] = {{'E', '+', 'T'}, {'T', 'b', '
   {'F', '10', '10'}
Char Stack[5][2];
int top = -1;
void instel (char pro Char re)
   int i;
   for(i=0; i<8; i++)
   {
    if(arr[i][0]=pro && arr[i][1] == re)
   {
}
```

```
int main(){
  int i=0,j;
  char pro,re,pri='\0';
  for(i=0;i<6;++i)
  {
    for(j=2;j>=0;--j)
    {
      if(res[i][j]=='+'||res[i][i]=='*'||res[i][j]=='c'||res[i][j]=='\0')
      {
        install(prod[i],res[i][j]);
        break;
      }
    }
```

Output:

E + f
E * f
E (f
E) f
E i f
E $ f
f + f
f * f
f (f
f) f
f $ f

Experiment-21

Aim: To write lex specification file to take cprogram from a c file count the no of Character line.

program:

```
#include <stdio.h>
int main()
{
int n1, n2, s;
printf("enter two integer:");
scanf("%d %d", &n1, &n2);
  s= n1+n2;
printf("%d %d= %d", n1, n2, s);
  return 0;
}
```

```
program (cl.l)
%{
   int nchar, nword, nline;
%}
%%
\n line++ inchar ++;
[^\t]+ {nword ++, nchar+= yleng;}
  { char ++}
%%
int ywrap (void){}
```

```
    return;
}
int main(int argc, char* argv[])
{
    yyin = fopen(argv[1], "r");
    yylex();
    printf("Number Character = %d", char);
    printf("No of word = %d", nword);
    fclose(yyin);
}
```

Output:

```
G:\lex>flex count.l
G:\lex>gcc lex.yy.c
No. of character = 233
No of word = 33
No of line = 0
```

Experiment-22

Aim: Write a lex programme to print all Constant in C program.

Program

```
#include <stdio.h>

void main()
{ int a,b,c=30;
  printf("hello")
program: (const Constant)
 digit [0-9]
%{ %
    int con=0
   %}
  %%
[digit]+ {cons++; printf("%s", yytext);
   \n}
  %%

int yywrap(void)
{
  return 1;
}

int main(void)
{
  FILE *f;
```

```
Char file[10];
Printf("enter file name");
Scanf("%s", file);
f = fopen(file, "r");
yyin = f;
yylex();
Printf("Num: %d", count);
fclose(yyin);
}
```

Output:

```
G = flex lex.l
G.lex > gcc lex.yy.c
G.lex > a.exe
3/4 is Constant
30 is Constant
```

Experiment -23

Aim: Write a LEX program to count the no of macros defined and header file included in c.

Program:

```
%{
int nm, nh;
%}
%%
^#define[ ]+[a-zA-Z]+ {nm++;}
^#include[ ]+[<"][a-zA-Z.]+[>"] {nh++;}
.|\n {}
%%
int yywrap(void){
 return 1;
}
int main(int argc, char *argv[])
{ yyin = fopen(argv[1],"r");
  yylex();
  printf("Number of macro = %d\n", nm);
  printf("No of header file = %d\n", nh);
}
```

Output:

```
G:\lex> flex Count_macro.l

G:\lex> gcc lex.yy.c

No of macros = 1
No of header file = 2
```

Experiment 24

Aim: To write a lex program to print all HTML tag in the tag file.

Program:

```
%{
int tag
%}
%%
'<"<^>]*>  {tag++;printf("%s\n",yytext);}
.|\n {}
%%
int yywrap(void)
{
  return 1;}
int main(void)
{
  FILE *f;
Char file[20];
printf("Enter file name:");
scanf("%s",file);
f=fopen(file,"r");
yyin=f;
yylex();
```

Output:

```
G:\lex>flex html.l
G:\lex>gcc lex.yy.c
G:\lex>a.exe
Enter file name: Simple html.
```

Experiment 25

Aim: Write a LEX program which add line no of given C program file and display the same in the Standard output.

Program:

```
%{
int yylineno;
%}
%%
^(.*)\n printf("%4d %s",++yylineno, yytext);
%%
int yywrap(void){
return 1
}
int main(int argc, char *argv[]){
yyin = fopen(argv[1],"r");
fclose(yyin)
}
```

Output:

```
G:\lex> flex add lines.l
G:\lex> gcc lex.yy.c
#define PI 3.14
#include<stdio.h>
#include<conio.h>
void main()
{
  int a,b,c=30;
  printf("hello")
}
```

Output

Experiment -26

Aim: write a LEX program to count the no. of comment line in a given C program and eliminate them and write into another file.

Program:

```
%{
int com=0;
%}
%s Comment
%%
"/*" BEGIN Comment;
<Comment>"*/"
<Comment>\n {com++;}
. \n {fprintf(yyout,"%s",yytext);}
%%
void main(int argc,char* argv[])
{ if(argc!=3)
{ printf("usage: a.exe input.c output.c\n");
  exit(0);
}
```

```
yyin = fopen (argv[1], "r");
yyout = fopen (argv[2], "w");
yylex();
printf("\n no of  Commit are = %d \n", com);
}
int yywrap()
{
  return 1;
}
```

Output:

```
$: \lex > flex Comment.l
$: \lex > gcc lex.yy.c
usage : a.exe input.c output.c
Number of Comments are = 2.
```

Experiment-27

Aim: Write a Lex program to identify the Capital the Capital word from the given input.

program:

```
%%
[A-Z]+[\t\n] {printf("%s is a CapitalWord\n",yytext);}
.;
%%
int main()
{ printf("Enter String:\n");
  yylex();
}
int yywrap()
{
  return 1;
}
```

Output: C:\Lex>a.exe

Enter a String:

CAPITAL of INDIA is DELHI

CAPITAL is CapitalWord

INDIA is CapitalWord

DELHI is CapitalWord.

Experiment - 28

Aim: Implement a LEX program to check mobile no valid or not

Program:

```
%%
[1-9][0-9]{9} { printf("Mobile no Valid\n"); }
. { printf("\n Mobile No is InValid\n"); }
%%
int main()
{ printf("Enter Number:")
  yylex();
  printf("\n");
  return 0;
}
```

Output:

```
C:\lex>a.exe
Enter Number: 7825726428
Mobile no valid.
```

Experiment - 29

Aim: Write a LEX program to convert the substring abc to ABC from the given input string.

Program:

```
%{
int i;
%}
%%
[a-z A-Z]* { for(i=0;i<yyleng;i++)
        {if ((yytext[i]=='a')&&(yytext[i+1]=='b')
         &&(yytext[i+2]=='c'))
{ yytext[i]='A';
  yytext[i+1]='B';
  yytext[i+2]='C';
 }
}
printf("%s",yytext);
}
[\t]* return;
.* {ECHO;}
\n printf("%s",yytext;}
%%
```

Experiment - 30

Aim: LEX program to recognise no of number and words in a statement

Program:

```
%%
[\t]+;
[0-9]+|[0-9]*\.[0-9]+ {printf("\n%s is Number",yytext);}
#.* {printf("\n%s is comment",yytext);}
[a-z A-Z]+ {printf("\n%s is word",yytext);}
\n {ECHO;}
%%

int main()
{
  while(yylex());
}
int yywrap()
{
  return;
}
```