Experiment - 1

Aim:
To develop lexical analyzer to identify constants operators using C program.

Algorithm:
1. Start the code and import the packages
2. Declare the variables
3. Assign the variables
4. Save and run the code.

Program:-

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
int main()
{ int i,rc=0,m, oc=0,j;
char b[30],operator[30], identifiers[30], constants[30];
printf("Enter the string: ");
for(i=0; i<strlen(b); i++)
{
  if(isspace(b[i]))
    {continue;
  }
  else if(isalpha(b[i]))
    {identifiers[rc] =b[i];
    rc++
```

```
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;
constants[cc]=m;
cc++;
}
else {
if(b[i]=="*")
{
operators[oc]="*";
oc++
}
else if(b[i]=='_')
{
operators[oc]='_';
oc++;
}
printf("Identifiers:");
for(j=0;j<c;j++)
{
printf("%c",identifiers[j]);
}
printf("\n constants:");
```

```
for(j=0;j<cc;j++)
{printf("%d",constants[j]);
}
printf("\n operators: ");
for(j=0;j<oc;j++)
{printf("%c",operators[j]);
}
}
```

Output:

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

identifiers: a b c

constants: 100

operators: = + * +

Result:-

A c program to identify constants, operators and identifiers is developed.

Experiment -2

Aim:

To develop a lexical analyzer to identify the given line is comment or not

Algorithm:-

* Start the code and import the packages
* Declare the variables
* Assign the variables,
* Save and Run the code.

Program:

```
#include<stdio.h>
#include<conio.h>
int main()
{
 char com[30];
 int i=2,a=0;
 printf("\n Enter Comment: ");
 gets(com);
 if(com[0]=='/')
  { if(com[1]=='/')
   { printf("\n It is a comment");
   }
  else if[com[1]=='*')
   { for(i=2; i<=30;i++
    {
```

```
If (com[i]=='*' && com[i+1]=='/')
{printf("\n It is a comment");
  a=1;
  break;
}
else {
   continue;
   }
If(a==0)
{ printf("\n It is not a comment");
else
   printf("\n It is not a comment");
}
```

Output:

I/P: Enter a comment : // hello

o/p: It is a comment

I/P : Enter a comment : hello

o/p : It is not a comment.

Result:

Using C program, we developed a lexical analyser to check if is a comment or not

Experiment No: 3

Aim:-

To design a lexical Analyzer to find the number of whitespaces and newline characters using C.

Algorithm:-

* Start the code and import the packages
* Declare the variables
* Assign the variables
* Save and run the code.

Program:-

```
#include<stdio.h>
int main()
{
char str[100];
int words = 0, newline = 0, characters = 0;
scanf("%[^\n]", str);
for(int i= 0; str[i] != '\0'; i++)
{ if (str[i] == ' ')
    { words++;
    }
else if (str[i] == '\n')
  { newline ++;
```

```
else if(str[i]!=' '&&str[i]!='\n'){
    characters++;
  }
}
if(characters>0)
  {words++;
   newline++;
  }
printf("Total no.of words:");
printf("Total no.of line: %d", newline);
printf("Total no.of characters: %d\n", characters);
return 0;
}
```

Output:

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

Total no.of words: 18
Total no.of line: 7

Result:

Hence the program is designed successfully.

Experiment No-4

Aim:

To develop a lexical analyzer to test whether a given identifier is valid or not using C

Algorithm:

* Start the code and import the packages
* Declare the variables
* Assign the variable
* Save and run the code.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>

int main()
()
{
  char a[10];
  int flag, i=1;
  printf("\n Enter an identifier");
  gets(a);
  while(a[i]!='\0');
      {
        if(isdigit(a[i])||isalpha(a[i]))
        { flag=0;
        } break;
```

```
    i++;
}
if(flag == 1)
  { printf("In valid Identifier");
}
```

Output:-

Enter an Identifier: abc123
valid Identifier

Result:

Hence, the lexical analyzer to test whether a given Identifier is valid or not is developed successfully.

Experiment No: 5

Aim:

To implement a C program to eliminate left recursion from a given CFG

$S \rightarrow (L)/a$

$L \rightarrow L,S/S$

Algorithm:

1. Start the code and import the packages
2. Declare the variables
3. Assign the variables.
4. Save and run the code

Program:

```
#include<stdio.h>
#include<string.h>
#define SIZE 10
int main()
{
  char non_terminal;
  char beta, alpha;
    int num;
  char product[10][SIZE];
  int index = 1;
  printf("Enter No. of Productions:");
  scanf("%d",&
```

```
printf("Enter the grammar as E->E-A:\n");
for(int i=0; i<num; i++)
  {
   scanf("%s",production[i]);
  }
for(int i=0; i<num; i++)
  {printf("GRAMMAR::: %s", production[i]);
   non_terminal = production[i][0];
   if(non_terminal==(production[i][index]))
      alpha = production[i][index+1];
     printf("is left recursive:\n");
   while(production[i][index]!=0 && production[i][!='|')
     index++;
   if(production[i][index]!=0)
     {beta= production[i][index+1];
     printf("Grammar without left recursion:\n");
     printf("%c->%c%c", nonterminal, beta,
                          nonterminal);
     printf("%c'->%c%c'|E", nonterminal, alpha,
                          nonterminal);
  .}
}
```

Output:-

Enter Number of Productions: 2

Enter the grammar as E → E-A:

S → (L) | a

L → L,S | S

GRAMMAR :: : S → (L) | a is not recursive

GRAMMAR :: : L → (L), S | S is left recursive

Grammar without left recursion:

L → SL'

L' → L' | ε

Result:-

Using C program, we eliminated left recursion from a given CFG.

Experiment No. 6

Aim:

To implement either top Down parsing technique or Bottom up parsing technique to check whether the given input string is satisfying the grammar or not

Algorithm:

1. Start the code and import the packages
2. Declare the variables
3. Assign the variables
4. Save and run the code

Program:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main(){
  char string[50];
   int flag, count=0;
   printf("the grammar is: S->aS, S->Sb, S->ab \n");
   printf("Enter the string to be checked:");
   gets(string);
   if(string[0]=='b')
      { flag=0;
```

```
for(count=1;string[count-1]!='\0';count++){
    if(string[count]=='b')
    { flag=1;
      continue;
    }
  else if((flag==1)&&(string[count]=='a')){
    printf("The string doesnot belong to the
        specified grammar");
      break;
  }
  else
   {
    printf("string accepted");
  }
 }
}
}
```

Output:

The grammar is: S→aA, S→Sb, S→ab

Enter the string to be checked : abb

String accepted.

Result:

Hence the program is executed successfully

Aim:-

To implement a lexical Analyzer to Scan and Count the number of characters, words and lines in a file.

Algorithm:-

1. Start the code and import the packages
2. Declare the variables
3. Assign the variables.
4. Save and run the code.

Program:-

```
#include<stdio.h>
int main()
{ char str[100];
  int words=0, newline=0, characters=0;
  scanf("%s",str);
  for(int i=0; str[i]!='\0'; i++)
  { if(str[i]==' ')
    { words++;
    }
    else if(str[i]=='\n')
    { newline++;
      words++;
  }
```

```
else if (str[i] != ' ' && str[i] != '\n')
    {
      characters++;
    }
  }
if (characters>0)
  {
    words++;
    newline++;
  }
printf("Total no. of words: %d", words);
printf("Total no. of line: %d, newline);
printf("Total no. of characters: %d", characters);
return 0;
}
```

Output:

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

Total no. of words: 18
Total no of lines: 7.

Result:

Lexical Analyzer to scan and count the no. of characters, words and lines in a file is executed successfully.

Aim:

To implement the back end of the compiler.

Algorithm:

1. Start the code and import the packages.
2. Declare the variable
3. Assign the variables
4. Save and run the code

Program:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{ int n,i,j;
  char a[50][50];
  printf("Enter the no: intermediate code:");
  scanf("%d",&n);
  for(i=0;i<n;i++)
  { printf("Enter the %d address code:",i+1);
    for(j=0;j<6;j++)
    { scanf("%c",&a[i][j]);
    }
}
printf("The generated code is:");
```

```
for(i=0;i<n;i++)
{
printf("mov%cR%d",a[i][5],i);
if(a[i][4]=='-')
{
printf("sub%cR%d",a[i][5],i);
}
if(a[i][4]=='+')
{
printf("add%cR%d",a[i][5],i);
}

if(a[i][4]=='*')

{
printf("mul%cR%d",a[i][5],i);
}
if(a[i][4]=='/')
{
printf("div%cR%d",a[i][5],i);
}
printf("mov R%d%c",1,a[i][1]);
printf("\n");
}
return 0;
}
```

Output:

Enter the no: intermediate code: 2

Enter the 3 address code 1: a=b+c

Enter the 3 address code 2: d=n*d

the generated code is

```
mov b,R0
add c,R0
mov R0,a
mov n,R1
mul d,R1
mov R1,d
```

Result:-

Back end of the compiler is implemented successfully.

Experiment No-9

Aim:

To design a lexical analyzer to validate operators to recognize the operators +, -, *, / using regular arithmetic operators using C.

* Start the code and import the packages
* Declare the variables.
* Assign the variables
* Save and Run the code.

```
#include<stdio.h>
#include<conio.h>
int main()
{
char s[5];
printf("\n Enter any operator: ");
get(s);
switch(s[10])
{
case '>':
if(s[1]=='=')
printf("\nGreater than (or) equal");
else
printf("\nGreater than");
break;
```

```
printf("In less than or equal");
else printf("In Equal to");
    break;
case '=':
    if(s[i]=='=')
      printf("In Equal to");
    else
      printf("In Bit Not");
      break;
  case '&';
      If(s[i]=='&')
          printf("Logical AND");
      else
        printf("In Bitwise AND");
        breaks
    case '|';
        If(s[i]=='|')
          printf("In logical or");
        else printf("In Bitwise OR");
          break:
    case '+':
          printf("In Addition");
            breaks
      case '-';
          printf("In Subtraction");
            break;
```

```
case '*':
        printf("\n multiplication");
        break;
case '%':
        printf("modulus");
        break;
default:
        printf("\n Not an Operator");
    }
}
```

Output:

Enter any operator: <=

Less than or equal

Result:

Design an Analyzer to identify the operators +, -, *, / using c program.

Experiment No 10:

Aim:

To write a C program to implement the operator precedence parsing.

Algorithm:

* Start the code and import packages
* Assign the variables
* Declare the variables
* Save and run the code.

Programming:

```
#include <string.h>
#include <stdio.h>
char *input;
int i=0;
char lasthandle[6], stack[50];
int top=0,l;
char prec[9][9] = {
int getindex(char c)
{ switch(c)
  { case '+': return 0;
    case '-': return 1;
    case '*': return 2;
    case '/': return 3;
```

```
        case 'n' : return 4;
        case '+' : return 5;
        case '(' : return 6;
        case ')' : return 7;
        case '$' : return 8;
    }
}
int shift()
{
    stack[++top] = *input++;
    stack[top+1] = '\0';
}
int reduce()
{ int i, len, found, t;
  if(stack[top] == handles[i][0] && top+1 >= len)
  {
    found = 1;
    for(t=0; t<len; t++)
  else
    printf("\n Not Accepted");
}
```

Output:

Enter the string

T*(T+T)*T

STACK	INPUT	ACTION
$T	*(T+T)*T$	shift
$E	*(T+T)*T$	Reduce: E→T
$E*	(T+T)*T$	shift
$E*(	T+T)*T$	shift
$E*(E	+T)*T$	shift
$E*(E+	T)*T$	shift
$E*(E+T	)*T$	shift
$E*(E	)*T$	shift
$E*(E)	*T$	shift
$E*	T$	shift
$E*T	$	shift
$E	$	shift

Result:

A program to implement the operator procedure is developed.

Write a LEX specification file to take input C program from a .c file and count the number of characters, number of lines & number of words.

Input Source program:-

```
#include<stdio.h>
int main()
{ int num1, num2, sum;
  printf("Enter two integers :");
  scanf("%d %d", &num1, &num2);
  sum = num1 + num2;
  printf("%d + %d = %d", num1, num2, sum);
}
```

Program:-

```
%{
int nchar, nword, nline;
%}
%%
\n {nline++; nchar++;}
[^ \t\n]+ {nword++, nchar += yyleng;}
. {nchar++;}
%%
```

```
}
int main(int argc, char *gr[])
{ yyin= fopen(argv[1], "r");
  yylex();
printf("No. of characters = %d\n", nchar);
  printf("No. of words = %d\n", nword);
  printf("No. of lines = %d\n", nline);
  fclose(yyin);
}
```

Output:-

```
flex count_line.l
gcc lex.yy.c
a.exe sample.c
No. of characters = 200
No. of words = 17
No. of lines = 10.
```

Result:-

A lex program for counting no. of characters is developed.

write a LEX program to print all the constants in the given C source program file

Input Source Program:-

```
#define P 314
#include<stdio.h>
#include<conio.h>
void main()
{ int a,b,c=10;
  printf("hello");
}
```

Program:-

```
digit [0-9]
%{
 int cons=0;
%}
%%
{digit}+ {cons++; printf("%s is constant\n", 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("No. of Constants: %d\n",cons);
fclose(yyin);
}
```

Output:-

```
flex countconstants(
gcc lex.yy.c
a.exe
Enter file Name: sample.c
314 is a constant
10 is a constant
No. of constants: 2
```

Result:-

A LEX program to print the all constants from the source file.

Experiment-14

Write a LEX program to count the number of macros defined and header files included in the C program.

Input Source Program:-

```
#define PI 3.14
# include <stdio.h>
#include<conio.h>
void main()
{ int a,b,c=10;
 printf("hellow");
}
```

Program:-

```
%{
int nmacro, nheader;
%}
%%
^#define{n macro++;}
^#include{n header++;}
-|\n{ }
%%
int yywrap (void)
    return 1;
}
int main(int argc, char *argv[])
  yyin= fopen(argv[1],"r");
```

```
yylex();
printf("No. of macros defined = %d\n", nmacro);
printf("No of header files included = %d\n", nhdr);
fclose(yyin);
}
```

Output:-

```
flex count_macro.l
gcc lex.yy.c
a.exe
No. of macros defined = 1
No. of header files included = 2
```

Result:-

A LEX program to count the no. of macros defined and header files included.

Experiment - 15

Write a LEX program to print all HTML tags in the input file.

Input Source Program:-

```
<html>
<body>
<h1> my first heading </h1>
<p> my first paragraph </p>
</body>
</html>
```

Program:-

```
%{
int tags;
%}
%%
<[^>]*>{tags++; printf("%s\n", yytext);}
.|\n {}
%%
int yywrap(void){
return 1; }
int main(void)
{
  FILE *f;
  char file[10];
printf("Enter file's Name: ");
scanf("%s", file);
```

```
f = fopen(file,"r");
yyin = f;
yylex();
printf("\n No.of html tags: %d", tags);
fclose(yyin);
}
```

Output:-

```
flex.html.l
gcc lex.yy.c
a.exe
Enter file Name: sample.html
<html>
<body>
<h1>
</h1>
</p>
</body>
</html>
No.of html tags: 8
```

Result:

A LEX program to print all HTML tags in a input file is developed.

Write a LEX program which made adds the numbers for the given C program file and display the same to the standard output

Input Source Program:-

```
#define PI 3.14
#include<stdio.h>
#include<conio.h>
 void main()
{
  int a,b,c=10;
   printf("hello");
}
```

Program:-

```
%{
int yylineno;
%}
%%
%%
int yywrap(void){
  return 1;
}
int main(int argc,char *argv[]){
yy in = fopen(argv[1],"r");
 yylex();
fclose(yyin);
}
```

Output :-

flex addlines.nos.l

gcc lex.yy.c

a.exe

```
1. #define PI 3.14
2. #include<stdio.h>
3. #include<conio.h>
4. void main()
   {int a,b,c=30;
    printf("hello");
   }
```

Result :-

A LEX program which adds line nos to the given C program file & display the same to the standard O/P

Write a LEX program to identify the capital words from the given input

Program:-

```
%%
[A-Z]+ [\t\n] {printf("%s is a capital word\n",yyte
%%
int main()
{printf("Enter string:\n");
    yylex;
}
int yywrap()
{ return 1;
}
```

Output:

```
CAPTIAL of INDIA IS DELHI
CAPTIAL is a capital word
INDIA is a capital word
DELHI IS a capital word
```

Result:-

A LEX program to identify the capital words from i/p is developed

Write a LEX program to check the email address is valid or not

Program:-

```
%{
int flag=0;
%}
%%
[a-z]+[0-9]+@[a-z]+".com"/".in"{flag=1;}
%%
int main()
{ yylex();
  if(flag==1)
    printf("Accepted");
  else
    printf("Not accepted");
}
int yywrap()
```

Output:-

sse123@gmail.com
Accepted.

Result

A LEX program to check email address is valid or not

Experiment no-20

Implement a LEX program to check whether the mobile number is valid or not

Program:-

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

Output:-

Enter mobile No: 7856451489
Mobile No. Valid

Result

A lex program to check whether mobile number is valid or not

Experiment no: 21

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);}
%%
int main()
{yylex();}
int yywrap()
{return1;
 }
```

Output

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

G:\lex>gcc lex.yy.c

G:\lex>a.exe
abcdefghabcijkla
ABCdefghABCijkla
G:\lex>
```

Result

A lex program to convert the substring is developed successfully

Write a LEX program to identify and count positive and negative numbers.

Program:-

```
%{
int positive_no=0, negative_no=0;
%}
%%
^[-][0-9]+{negative_no++;
        printf("negative number = %s\n",
            yytext);}

[0-9]+{positive_no++;
        printf("positive number = %s\n")

  return 0;
}
```

Output:-

```
-10
negative number = -10

20
positive number = 20
number of positive numbers = 1, number of negative = 1
```

Result

A lex program to count positive and negative numbers created successfully.

Experiment no: 23

Lex Program to validate the URL

Program: (url.l)

```
%%
(http)|(ftp)s?://[a-zA-Z0-9]([a-z]+[a-zA-Z0-9]+
        *(printf("\n URL Valid\n");}.+
        (printf("\n URL Invalid\n");}
%%
void main()
{ printf("\n Enter URL: ");
  yylex();
  printf("\n");
}
int yywrap()
{}
```

Output

Enter URL: https://www.sse.in

URL Invalid

https://www.sse.in

URL valid

Result

A lex program to validate URL is developed successfully.

Experiment no: 24

Write a LEX program to validate DOB of students.

Program: (dob.l)

```
%%
((0[1-9])|[1-2][0-9])|(3[0-1]))|((0[1-9])|(1[0-2]))
                    Printf("valid DOB");
%%
int main()
{ yylex();
  return 0;
}
int yywrap()
{}
```

Output:

26/07/1995

Valid DOB.

Result:

A lex program to validate DOB of students is successfully developed.

Write a LEX program to check whether the given input is digit or not

Program:-

```
%%
[0-9]+ {printf("\n Valid digit \n");}
.* printf("\n Invalid digit \n");
%%
int yywrap(){}
int main()
 {yylex();
  return 0;
}
```

Output

```
25
Valid digit

hi6
Invalid digit
```

Result:-

A lex program to check whether the given input is digit or not is successfully developed.

Experiment No: 26

Write a LEX program to count the number of macros defined and header files included in the C program.

Program:

```
%{
int nmacro, nheader;
%}
%%
^define {nmacro++;}
^#include {nheader++;}
.|\n {}
%%
int yywrap(void){
  return 1;
}
int main(int argc, char *argv[]){
yyin=fopen(argv[1], "r");
  yylex();
printf("Number of macros defined=%d\n", nmacro);
printf("Number of header files included=%d\n",
    nheader);
  fclose(yyin);
}
```

Output

Number of macros defined = 1

Number of header files included = 2

Result

A LEX program to count the number of macros defined and header files included in the C program.

Experiment No 27

Write a LEX program which adds line numbers to the given C program file and display the same in the standard output

Program:

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

Output:

```
#define PI 3.14

#include<stdio.h>

#include<conio.h>

void main()

{ int a,b,c = 30;

  printf("hello;"
```

Result

A LEX program which adds line numbers to the given C program file and display the same to the standard output.

Write a LEX program to count the number of comment lines in a given C program and eliminate them and write into another file.

Program:

```
%{
int com=0;
%}
%s COMMENT
%%
"/*" {BEGIN COMMENT;}
<COMMENT> "*/" {BEGIN 0; com++;}
<COMMENT> \n {com++;}
<COMMENT> . {;}
\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 number of comments are=%d\n", com);
}
```

```
int yywrap()
{
return 1;
}
```

Output

```
usage: a.exe input.c output.c
number of comment arc=2
```

Result

A lex program is developed successfully to count the number of comment lines

Experiment no: 10

Write a lexical Analyzer using FLEX. The program should seperate the tokens in the given c program and display with appropriate caption.

Program:-

```
digit [0-9]
letter [A-Za-z]
%{
int count_id, count_key;
%}
%%
(stdio.h|conio.h) {printf("%s is a standard library\n", yytext);}
(include|word|main|print|int) {printf("%s is a keyword
{letter}{letter}|{digit}{ printf("%s is a
identifier, yytext);
{digit}+ {printf("%s is a number, yytext );}
\"(.|[\n\t])*\" {printf("%s is a string literal\n", yytext);}
.|\n{}
%%
int yywrap(void){
return 1;
}
```

```
int main(int argc, char *argv[]);
yyin = fopen(argv[1], 'r');
  yylex();
  printf("number of identifiers = %d\n", count
printf("number of keywords = %d\n", count_key
fclose(yyin);
}
```

Output:-

include is keyword
stdio.h is a standard library
void is a keyword
main is a keyword
int is a keyword
a is a identifier
b is a identifier
c is a identifier
10 is a number
printf is a keyword
"hello" is a string literal
number of identifiers = 3
number of keywords = 5

Result

A lexical analyzer is developed succes