EXERCISE 2: IMPLEMENTATION OF LEXICAL ANALYZER USING LEX (Token Generation)

AIM: To perform token separation by writing patterns and actions using LEX.

ALGORITHM:

1. Declare the necessary symbols and header files needed.

2. Write the pattern and corresponding actions to be taken

    2.1. if the pattern is #,* print it as a header file
    2.2. if the pattern is int|float|char|return|main|… print it as a keyword
    2.3. if the pattern is [“][a-zA-z][“] print it as a string
    2.4. Similarly identify patterns and actions for all tokens.

3. In the main function declare a file pointer to read input file and call yylex() to find matching patterns and corresponding actions to be taken.

CODE:

```
%option noyywrap
%{
    #include<stdio.h>
    void yyerror(char *);
%}
letter [a-zA-Z]
digit [0-9]
op [-+*/]
punct [,.;"]
%%
else|if|void|int  {printf("%s is a keyword",yytext);}
{digit}+  {printf("%s is a number",yytext);}
{letter}({letter}|{digit})*   {printf("%s is an identifier",yytext);}
{op}  {printf("%s is an operator",yytext);}
[ ]  ;
\)  {printf("%s is close parenthesis",yytext);}
{punct}  {printf("%s is a punctuation",yytext);}
.  yyerror("error");
%%
void yyerror(char *s)
{
fprintf(stderr,"%s\n",s);
}
int main(int argc, char *argv[])
{
    FILE *fp;

    if((fp=fopen(argv[1],"r"))==NULL)
    {printf("file does not exist");}
    yyin=fp;
    yylex();
    return 0;
}
```

Result :

Ex.3 Evaluation of Arithmetic expression using Ambiguous Grammar(Use Lex and Yacc Tool)

E-> E+E | E-E|E*E | E/E| (E) | id

Lex file:

```
%option noyywrap
%{
    #include<stdio.h>
    #include"y.tab.h"
    void yyerror(char *s);
    extern int yylval;
%}
digit [0-9]
%%
{digit}+    {yylval=atoi(yytext);return NUM;}
[-+*/\n]    {return *yytext;}
\(          {return *yytext;}
\)          {return *yytext;}
.           {yyerror("syntax error");}
%%
```

YACC file:

```
%{
    #include<stdio.h>
    void yyerror(char*);
    extern int yylex(void);
%}
%token NUM
%%
S:
S E '\n'        {printf("%d\n",$2);}
|
;
E:
E '+' E             {$$=$1+$3;}
|E '-' E      {$$=$1-$3;}
|E '*' E      {$$=$1*$3;}
|E '/' E      {$$=$1/$3;}
|'(' E ')'    {$$=$2;}
|NUM              {$$=$1;}
%%
void yyerror(char *s)
{
printf("%s",s);
}
```

```
int main()
{
yyparse();
return 0;
}
```

Ex.4 Evaluation of Arithmetic expression using Unmbiguous Grammar(Use Lex and Yacc Tool)

E-> E+T | E-T|T
T->T*F | T/F|F
F-> (E) | id

Lex File:

```
%option noyywrap
%{
    #include<stdio.h>
    #include"y.tab.h"
    void yyerror(char *s);
    extern int yylval;
%}
digit [0-9]
%%
{digit}+    {yylval=atoi(yytext);return NUM;}
[-+*/\n]    {return *yytext;}
\(          {return *yytext;}
\)          {return *yytext;}
.           {yyerror("syntax error");}
%%
```

YACC file:

```
%{
    #include<stdio.h>
    void yyerror(char*);
    extern int yylex(void);
%}
%token NUM
%%
S:
S E '\n'        {printf("%d\n",$2);}
|
;
E:
E '+' T             {$$=$1+$3;}
|E '-' T      {$$=$1-$3;}
|T            {$$=$1;}
T:
```

```
T '*' F             {$$=$1*$3;}
|T '/' F      {$$=$1/$3;}
|F            {$$=$1;}
F:
'(' E ')'     {$$=$2;}
|NUM              {$$=$1;}
%%
void yyerror(char *s)
{
printf("%s",s);
}
int main()
{
yyparse();
return 0;
}
```

Ex.5 Use LEX and YACC tool to implement Desktop Calculator.

E-> E+T | E-T|T
T->T*F | T/F|F
F-> (E) | id

Lex File:

```
%option noyywrap
%{
    #include<stdio.h>
    #include"y.tab.h"
    void yyerror(char *s);
    extern int yylval;
%}
digit [0-9]
%%
{digit}+            {yylval=atoi(yytext);return NUM;}
[a-z]               {yylval=toascii(*yytext)-97;return ID;}
[A-Z]               {yylval=toascii(*yytext)-65;return ID;}
[-+*/=\n]           {return *yytext;}
\(                  {return *yytext;}
\)                  {return *yytext;}
.                   {yyerror("syntax error");}
%%
```

YACC file:

```
%{
    #include<stdio.h>
    void yyerror(char*);
    extern int yylex(void);
```

```
    int val[26];
%}
%token NUM ID
%%

S:
S E '\n'            {printf("%d\n",$2);}
| S ID '=' E '\n'    {val[$2]=$4;}
|
;
E:
E '+' T             {$$=$1+$3;}
|E '-' T            {$$=$1-$3;}
|T                  {$$=$1;}
T:
T '*' F             {$$=$1*$3;}
|T '/' F            {$$=$1/$3;}
|F                  {$$=$1;}
F:
'(' E ')'           {$$=$2;}
|NUM                {$$=$1;}
|ID                 {$$=val[$1];}
%%
void yyerror(char *s)
{
printf("%s",s);
}
int main()
{
yyparse();
return 0;
}
```

Ex. No. 6 RECURSIVE DESCENT PARSING

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int i=0 ,f=0;
char str[30];
void E();
void Eprime();
void T();
void Tprime();
void F();
```

```
void E()
{
printf("\nE->TE'");
T();
Eprime();
}
void Eprime()
{
if(str[i]=='+')
{
        printf("\n\E'->+TE'");
        i++;
        T();
        Eprime();
        }
else if((str[i]==')')||(str[i]=='$'))
        printf("\nE'->^");
}
void T()
{
printf("\nT->FT'");
F();
Tprime();
}
void Tprime()
{
        if(str[i]=='*')
                {
                printf("\nT'->*FT'");
        i++;
                F();
                Tprime();
                }
        else if((str[i]==')')||(str[i]=='+')||(str[i]=='$'))
                printf("\nT'->^");
}
void F()
{
        if(str[i]=='a')
        {
                printf("\nF->a");
```

```
                i++;
        }
        else if(str[i]=='(')
        {
                printf("\nF->(E)");
                i++;
                E();
                if(str[i]==')')
                        i++;
        }
        else
                f=1;
}
void main()
{
        int len;
        clrscr();
        printf("Enter the string: ");
        scanf("%s",str);
        len=strlen(str);
        str[len]='$';
        E();
        if((str[i]=='$')&&(f==0))
        printf("\nStringsucessfully parsed!");
        else
                printf("\nSyntax Error!");
getch();
}
```

Output 1

Enter the string: a+a*a

E->TE'
T->FT'
F->a
T'->^
E'->+TE'
T->FT'
F->a
T'->*FT'
F->a

T'->^
E'->^
String Sucessfully parsed!

Output 2
Enter the string: a++

E->TE'
T->FT'
F->a
T'->^
E'->+TE'
T->FT'
T'->^
E'->+TE'
T->FT'
T'->^
E'->^
Syntax Error!

Ex.No. 7-SHIFT REDUCE PARSER

Source code:

```
#include<stdio.h>
#include<string.h>
int  z,i,j,c;
char a[16],stk[15];
void reduce();
void main()
  {
puts("GRAMMAR is E->E+E \n E->E*E \n E->(E) \n E->a");
puts("enter input string ");
     gets(a);
     c=strlen(a);
     a[c]='$';
stk[0]='$';
puts("stack \t input \t action");
for(i=1,j=0;j<c; i++,j++)
     {
          if(a[j]=='a')
            {
          stk[i]=a[j];
          stk[i+1]='\0';
```

```
            a[j]=' ';
          printf("\n%s\t\t%s\tshift->a",stk,a);
          reduce();
            }
           else
            {
          stk[i]=a[j];
          stk[i+1]='\0';
            a[j]=' ';
          printf("\n%s\t\t%s\tshift->%c",stk,a,stk[i]);
          reduce();
            }
     }
            if(a[j]=='$')
          reduce();
            if((stk[1]=='E')&&(stk[2]=='\0'))
          printf("\n%s\t\t%s\tAccept",stk,a);
            else
          printf("\n%s\t\t%s\terror",stk,a);

  }

void reduce()
  {
for(z=1; z<=c; z++)
      if(stk[z]=='a')
           {
          stk[z]='E';
          stk[z+1]='\0';
          printf("\n%s\t\t%s\tReduce by E->a",stk,a);
           }
for(z=1; z<=c; z++)
     if(stk[z]=='E' &&stk[z+1]=='+' &&stk[z+2]=='E')
           {
          stk[z]='E';
          stk[z+1]='\0';
          stk[z+2]='\0';
          printf("\n%s\t\t%s\tReduce by E->E+E",stk,a);
          i=i-2;
           }
for(z=1; z<=c; z++)
    if(stk[z]=='E' &&stk[z+1]=='*' &&stk[z+2]=='E')
           {
          stk[z]='E';
          stk[z+1]='\0';
          stk[z+2]='\0';
```

```
        printf("\n%s\t\t%s\tReduce by E->E*E",stk,a);
        i=i-2;
         }
for(z=1; z<=c; z++)
    if(stk[z]=='(' &&stk[z+1]=='E' &&stk[z+2]==')')
         {
        stk[z]='E';
        stk[z+1]='\0';
        stk[z+2]='\0';
        printf("\n%s\t\t%s\tReduce by E->(E)",stk,a);
        i=i-2;
         }
  }
```

Output 1:

```
GRAMMAR is E->E+E
 E->E*E
 E->(E)
 E->a
enter input string
a+a*a
stack          input     action

$a             +a*a$ shift->a
$E             +a*a$ Reduce by E->a
$E+             a*a$  shift->+
$E+a             *a$  shift->a
$E+E             *a$  Reduce by E->a
$E               *a$  Reduce by E->E+E
$E*               a$  shift->*
$E*a              $   shift->a
$E*E              $   Reduce by E->a
$E                $   Reduce by E->E*E
$E                $   Accept
```

Ex. No. 8 Implement Operator Precedence Parser algorithm.

Algorithm:

Initialize: Set *ip* to point to the first symbol of *w*$
repeat:

Let *X* be the top stack symbol, and ***a*** the symbol pointed to by *ip*

```
            if $ is on the top of the stack and ip points to $
            then return
             else
               Let a be the top terminal on the stack, and b the symbol pointed to by ip
                  if a <· b or a =· b then
                                push b onto the stack
                                advance ip to the next input symbol
                  else if a ·> b then
repeat
                                  pop the stack
                                until the top stack terminal is related by <·
                                          to the terminal most recently popped
                           else error()
end
```

Source code:

```
#include<stdio.h>
#include<conio.h>
void main()
{
        char stack[20],ip[20],opt[10][10][1],ter[10];
        int i,j,k,n,top=0,col,row;
        clrscr();
        for(i=0;i<3;i++)
        {
          stack[i]=NULL;
          ip[i]=NULL;
          for(j=0;j<3;j++)
          {
            opt[i][j][0]=NULL;
          }
        }
        printf("Enter the no.of terminals:");
        scanf("%d",&n);
        printf("\nEnter the terminals:");
        scanf(" %s",ter);
        printf("\nEnter the table values:\n");
        for(i=0;i<n;i++)
        {
        for(j=0;j<n;j++)
         {
          printf("Enter the value for %c %c:",ter[i],ter[j]);
          scanf(" %s",opt[i][j]);
         }
        }
```

```
printf("\nOPERATOR PRECEDENCE TABLE:\n");
for(i=0;i<n;i++){printf("\t%c",ter[i]);}
printf("\n");
for(i=0;i<n;i++)
{
   printf("\n%c",ter[i]);
   for(j=0;j<n;j++)
   {
            printf("\t%c",opt[i][j][0]);

    }
 }
 stack[top]='$';
 printf("\nEnter the input string:");
 scanf(" %s",ip);
 i=0;
 printf("\nSTACK\t\tINPUT STRING\t\tACTION\n");
 printf("\n%s\t\t\t%s\t\t\t",stack,ip);
 while(i<=strlen(ip))
{
            for(k=0;k<n;k++)
            {
              if(stack[top]==ter[k])
              row=k;
              if(ip[i]==ter[k])
              col=k;
            }
            if((stack[top]=='$')&&(ip[i]=='$'))
            {
            printf("String is accepted");
            break;
            }
            else if((opt[row][col][0]=='<') ||(opt[row][col][0]=='='))
            {
              stack[++top]=opt[row][col][0];
              stack[++top]=ip[i];
              printf("Shift %c",ip[i]);
              i++;
            }
             else
             {
              if(opt[row][col][0]=='>')
              {
                while(stack[top]!='<')
                --top;
                top=top-1;
```

```
                printf("Reduce");
               }
              else
               {
                printf("\nString is not accepted");
                break;
               }
             }
 printf("\n");
 for(k=0;k<=top;k++)
 printf("%c",stack[k]);
 printf("\t\t\t");
 for(k=i;k<strlen(ip);k++)
 printf("%c",ip[k]);
 printf("\t\t\t");
}
getch();
}
```

Output 1:

Enter the no.of terminals:3

Enter the terminals:a+$

Enter the table values:
Enter the value for a a:e
Enter the value for a +:>
Enter the value for a $:>
Enter the value for + a:<
Enter the value for + +:>
Enter the value for + $:>
Enter the value for $ a:<
Enter the value for $ +:<
Enter the value for $ $:A

OPERATOR PRECEDENCE TABLE:

	a	+	$
a	e	>	>
+	<	>	>
$	<	<	A

Enter the input string:a+a$

STACK INPUT STRING ACTION

$	a+a$	Shift a
$<a	+a$	Reduce
$	+a$	Shift +
$<+	a$	Shift a
$<+<a	$	Reduce
$<+	$	Reduce
$	$	String is accepted

Exno 8 Implement the backend of the compiler to produce three address code generation

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>

struct three
{
char data[10],temp[7];
}s[30];
void main()
{
char d1[7],d2[7]="t";
int i=0,j=1,len=0;
FILE *f1,*f2;
clrscr();
f1=fopen("sum.txt","r");
f2=fopen("out.txt","w");
while(fscanf(f1,"%s",s[len].data)!=EOF)
len++;
itoa(j,d1,7);
strcat(d2,d1);
strcpy(s[j].temp,d2);
strcpy(d1,"");
strcpy(d2,"t");
if(!strcmp(s[3].data,"+"))
{
fprintf(f2,"%s=%s+%s",s[j].temp,s[i+2].data,s[i+4].data);
j++;
}
else if(!strcmp(s[3].data,"-"))
{
fprintf(f2,"%s=%s-%s",s[j].temp,s[i+2].data,s[i+4].data);
j++;
}
for(i=4;i<len-2;i+=2)
{
itoa(j,d1,7);
strcat(d2,d1);
```

```
strcpy(s[j].temp,d2);
if(!strcmp(s[i+1].data,"+"))
fprintf(f2,"\n%s=%s+%s",s[j].temp,s[j-1].temp,s[i+2].data);
else if(!strcmp(s[i+1].data,"-"))
fprintf(f2,"\n%s=%s-%s",s[j].temp,s[j-1].temp,s[i+2].data);
strcpy(d1,"");
strcpy(d2,"t");
j++;
}
fprintf(f2,"\n%s=%s",s[0].data,s[j-1].temp);
fclose(f1);
fclose(f2);
getch();
}
```

Input: sum.txt

```
out = in1 + in2 + in3 - in4
```

Output : out.txt

```
t1=in1+in2
t2=t1+in3
t3=t2-in4
out=t3
```

Exno 9 Symbol Table

ALGORITHM:

Start the program for performing insert, display, delete, search and modify option in symbol table
Define the structure of the Symbol Table
Enter the choice for performing the operations in the symbol Table
If the entered choice is 1, search the symbol table for the symbol to be inserted. If the symbol is
already present, it displays “Duplicate Symbol”. Else, insert the symbol and the corresponding address in
the symbol table.
If the entered choice is 2, the symbols present in the symbol table are displayed.

If the entered choice is 3, the symbol to be deleted is searched in the symbol table. If it is not found in the symbol table it displays “Label Not found”. Else, the symbol is deleted.
If the entered choice is 5, the symbol to be modified is searched in the symbol table.

Program

```
#include<stdio.h>
#include<ctype.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
void main()
{
 int i=0,j=0,x=0,n;
 void *p,*add[5];
 char ch,srch,b[15],d[15],c;
 printf("Expression terminated by $:");
 while((c=getchar())!='$')
 {
  b[i]=c;
  i++;
 }
 n=i-1;
 printf("Given Expression:");
 i=0;
 while(i<=n)
 {
  printf("%c",b[i]);
  i++;
 }
 printf("\n Symbol Table\n");
 printf("Symbol \t addr \t type");
 while(j<=n)
 {
  c=b[j];
  if(isalpha(toascii(c)))
  {
   p=malloc(c);
   add[x]=p;
   d[x]=c;
```

```
   printf("\n%c \t %d \t identifier\n",c,p);
   x++;
   j++;
  }
  else
  {
   ch=c;
   if(ch=='+'||ch=='-'||ch=='*'||ch=='=')
   {
    p=malloc(ch);
    add[x]=p;
    d[x]=ch;
    printf("\n %c \t %d \t operator\n",ch,p);
    x++;
    j++;
   }}}}
```

Input :

Expression terminated by $:A+B=C $

Output:

Given Expression:A+B=C

Symbol Table

Symbol	addr	type
A	22014656	identifier
+	22014736	operator
B	22014800	identifier
=	22014880	operator
C	22014960	identifier

Exno 11 Implementation of code optimization Techniques

```
#include<stdio.h>
#include<string.h>
struct op
{
char l;
char r[20];
}
op[10],pr[10];
void main()
{
int a,i,k,j,n,z=0,m,q;
char *p,*l;
char temp,t;
char *tem;
printf("Enter the Number of Values:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("left: ");
scanf(" %c",&op[i].l);
printf("right: ");
scanf(" %s",&op[i].r);
}
printf("Intermediate Code\n") ;
for(i=0;i<n;i++)
{
printf("%c=",op[i].l);
printf("%s\n",op[i].r);
}
for(i=0;i<n-1;i++)
{
temp=op[i].l;
for(j=0;j<n;j++)
{
p=strchr(op[j].r,temp);
if(p)
{
pr[z].l=op[i].l;
strcpy(pr[z].r,op[i].
```

```
r);
z++;
}
}
}
pr[z].l=op[n-1].l;
strcpy(pr[z].r,op[n-1].r);
z++;
printf("\nAfter Dead Code Elimination\n");
for(k=0;k<z;k++)
{
printf("%c\t=",pr[k].l);
printf("%s\n",pr[k].r);
}
for(m=0;m<z;m++)
{
tem=pr[m].r;
for(j=m+1;j<z;j++)
{
p=strstr(tem,pr[j].r);
if(p)
{
t=pr[j].l;
pr[j].l=pr[m].l;
for(i=0;i<z;i++)
{
l=strchr(pr[i].r,t) ;
if(l)
{
a=l-pr[i].r;
printf("pos: %d\n",a);
pr[i].r[a]=pr[m].l;
}}}}}
printf("Eliminate Common Expression\n");
for(i=0;i<z;i++)
{
printf("%c\t=",pr[i].l);
printf("%s\n",pr[i].r);
}
for(i=0;i<z;i++)
{
for(j=i+1;j<z;j++)
```

```
{
q=strcmp(pr[i].r,pr[j].r);
if((pr[i].l==pr[j].l)&&!q)
{
pr[i].l='\0';
}
}
}
printf("Optimized Code\n");
for(i=0;i<z;i++)
{
if(pr[i].l!='\0')
{
printf("%c=",pr[i].l);
printf("%s\n",pr[i].r);
}
}
}
```

```
INPUT & OUTPUT:
Enter the Number of Values:5
left: a right: 9
left: b right: c+d
left: e right: c+d
left: f right: b+e
left: r right: f
Intermediate Code
a=9
b=c+d
e=c+d
f=b+e
r=f

After Dead Code Elimination nbt=c+dnet=c+dnft=b+enrt=fnpos:
Eliminate Common Expression
b =c+d
b =c+d
f =b+b
r =f
Optimized Code
b=c+d
```

```
f=b+b

r=f
```