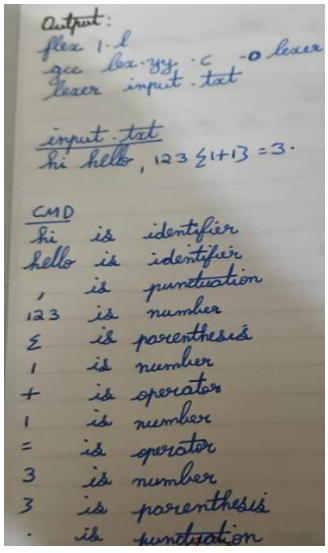
CD PGMS

2. IMPLEMENTATION OF LEXICAL ANALYZER USING LEX (Token Generation)

2.l

```
%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;}
```

OUTPUT:



3. EVALUATION OF ARITHMETIC EXPRESSION USING YACC AND LEX(Ambiguous Grammar)

3.l

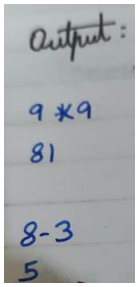
```
%option noyywrap
%{ #include<stdio.h>
#include"y.tab.h"
void yyerror(char *s);
extern int yylval;
%}
digit [0-9]
%%
{digit}+ {yylval=atoi(yytext);return NUM;}
[-+*/\n] {return *yytext;}
\( {return *yytext;}
\) {return *yytext;}
. {yyerror("syntax error");}
%%
```

3.y

```
%{ #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; }
```

OUTPUT:



4. EVALUATION OF ARITHMETIC EXPRESSION USING YACC AND LEX (Unambiguous Grammar)

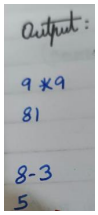
4.l

```
%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");}
%%
```

4.y

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

OUTPUT:



Ex.5 Implement Desktop Calculator

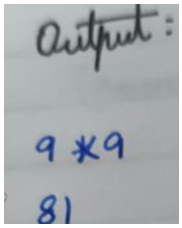
5.l

```
%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");}
%%
```

5.y

```
%{ #include<stdio.h>
void yyerror(char*);
extern int yylex(void);
int val[26];%}
%token NUM %%
S:
S E '\n' {printf("%d\n",$2);}
|S |D ‘=’ E ‘\n’ {val[$2]=$4;}
| ; E:
E '+' T {$$=$1+$3;}
|E '-' T {$$=$1-$3;}
|T {{$$=$1;}
T:|T '*' F {$$=$1*$3;}
|T '/' F {$$=$1/$3;}
|F
F:'(' E ')' {$$=$2;}
|NUM {$$=$1;}%%
void yyerror(char *s)
{ printf("%s",s); }
int main()
{ yyparse(); return 0; }
```

OUTPUT:



6.IMPLEMENT RECURSIVE DESCENT PARSER

```
#include <stdio.h>
#include <string.h>
int i = 0, f = 0;
char str[30];
void E(), Eprime(), T(), Tprime(), F();
void E(){printf("\nE->TE'"); T(); Eprime();}
void Eprime()
{if (str[i]=='+'){printf("\nE'->+TE'");i++;T();Eprime();}}
void T()
{printf("\nT->FT'");F();Tprime();}
void Tprime()
{if(str[i]=='*')
{printf("\nT'->*FT'");i++;F();Tprime();}}
void F()
{if (str[i] == 'a'){printf("\nF->a");i++;}
else if(str[i] =='('){printf("\nF->(E)");i++;E();if(str[i]==')')i++;f = 1; }
else f = 1; }
int main() {
printf("Enter the string: ");
scanf("%29s", str);
str[strlen(str)] = '$';
E();
printf((str[i] == '$' && !f) ? "\nString parsed!" : "\nSyntax Error!");
return 0;}
```

OUTPUT:

```
Enter the string: a+a*a
E->TE'
T->FT'
F->a
E'->+TE'
T->FT'
F->a
T'->*FT'
F->a
String parsed!
```

7. IMPLEMENT SHIFT REDUCE PARSER

```
#include <stdio.h>
#include <string.h>
int i = 0, j = 0, c;
char a[16], stk[15] = "$";
void reduce()
{for (int z = 1; z <= c; z++)
{if (stk[z] == 'a')
{stk[z] = 'E'; printf("\n%s\t%s\tReduce: E->a", stk, a);}
if (stk[z] == 'E' && stk[z + 1] == '+' && stk[z + 2] == 'E')
{stk[z + 1]=stk[z+2]='\0'; printf("\n%s\t%s\tReduce: E->E+E",stk,a);i-=2;}
if (stk[z] == 'E' && stk[z + 1] == '*' && stk[z + 2] == 'E')
{stk[z+1]=stk[z+2]='\0';printf("\n%s\t%s\tReduce: E->E*E",stk,a);i-=2;}
if (stk[z] == '(' && stk[z + 1] == 'E' && stk[z + 2] == ')')
{stk[z]='E';stk[z+1]=stk[z+2]='\0';printf("\n%s\t%s\tReduce: E->(E)",stk,a);
i-= 2;}
}
}
int main()
{printf("GRAMMAR: E->E+E | E->E*E | E->(E) | E->a\nEnter input: ");
fgets(a, sizeof(a), stdin);
a[strcspn(a, "\n")] = '\0';
c = strlen(a);
a[c] = '$';
printf("Stack\tInput\tAction");
for (i = 1; j < c; i++, j++) {
stk[i] = a[j], stk[i + 1] = '\0', a[j] = ' ';
printf("\n%s\t%s\tShift->%c", stk, a, stk[i]);
reduce();}
if (stk[1] == 'E' && stk[2] == '\0') printf("\n%s\t%s\tAccept", stk, a);
else printf("\n%s\t%s\tError", stk, a);
return 0;}
```

OUTPUT:

```
GRAMMAR: E->E+E | E->E*E | E->(E) | E->a
Enter input: a+a*a
Stack    Input    Action
$a        +a*a$   Shift->a
$E        +a*a$   Reduce: E->a
$E+        a*a$   Shift->+
$E+a        *a$   Shift->a
$E+E        *a$   Reduce: E->a
$E+E*        a$   Shift->*
$E           a$   Reduce: E->E+E
$E            $   Shift->a
$E            $   Reduce: E->a
$E            $   Accept
```

8. IMPLEMENT OPERATOR PRECEDENCE PARSER

```
#include<stdio.h>
#include <string.h>
void main()
{
char stack[20],ip[20],opt[10][10][1],ter[10];
int i,j,k,n,top=0,col,row;
for(i=0;i<3;i++)
{stack[i]='\0';
ip[i]='\0';
for(j=0;j<3;j++)
{ opt[i][j][0]='\0'; } }
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"); } }
```

OUTPUT:

```
Enter the no.of terminals:3
Enter the terminals:a+$
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
```

9. IMPLEMENT THE BACKEND OF THE COMPILER TO PRODUCE THREE ADDRESS CODE GENERATION

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct three {char data[10], temp[7];} s[30];
int main()
{FILE *f1 = fopen("sum.txt", "r"), *f2 = fopen("out.txt", "w");
int len = 0, j = 1; char d2[7] = "t";
while (fscanf(f1, "%s", s[len].data) != EOF) len++;
sprintf(s[j].temp, "t%d", j);
fprintf(f2, "%s=%s%s%s", s[j].temp, s[2].data, s[3].data, s[4].data);
for (int i = 4; i < len - 2; i += 2)
{sprintf(s[++j].temp, "t%d", j);
fprintf(f2, "\n%s=%s%s%s",s[j].temp,s[j-1].temp,s[i+1].data,s[i+2].data);}
fprintf(f2, "\n%s=%s", s[0].data, s[j].temp);
fclose(f1);
fclose(f2);
return 0;}
```

OUTPUT:

```
Input: sum.txt
out = in1 + in2 + in3 - in4
Output: out.txt
t1=in1+in2
t2=t1+in3
t3=t2-in4
out=t3
```

10. IMPLEMENT SYMBOL TABLE

```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
int main()
{char b[15], c;
void *add[5];
int i = 0, x = 0;
printf("Expression terminated by $: ");
while ((c = getchar()) != '$') b[i++] = c;
printf("Given Expression: %.*s\nSymbol\tAddr\tType\n", i, b);
for (int j = 0; j < i; j++)
{c = b[j];
if (isalpha(c) || strchr("+-*=", c))
{add[x] = malloc(1);
printf("%c\t%p\t%s\n", c, add[x], isalpha(c) ? "Identifier" : "Operator");
x++;}}
return 0;}
```

OUTPUT:

Expression terminated by $: a-c+d=d$

Given Expression: a-c+d=d

Symbol	Addr	Type
a	0x1117dac0	Identifier
-	0x1117dae0	Operator
c	0x1117db00	Identifier
+	0x1117db20	Operator
d	0x1117db40	Identifier
=	0x1117db60	Operator
d	0x1117db80	Identifier

11. IMPLEMENTATION OF CODE OPTIMIZATION TECHNIQUES

```
#include <stdio.h>
#include <string.h>
struct op
{char l, r[20];}op[10], pr[10];
int main()
{int n, i, j, z = 0;
char *p, t;
printf("Enter 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("\nIntermediate Code\n");
for (i = 0; i < n; i++) printf("%c=%s\n", op[i].l, op[i].r);
for (i = 0; i < n; i++)
{for (j = 0; j < n; j++)
if (strchr(op[j].r, op[i].l)) pr[z++] = op[i];}
pr[z++] = op[n - 1];
printf("\nAfter Dead Code Elimination\n");
for (i = 0; i < z; i++) printf("%c=%s\n", pr[i].l, pr[i].r);
for (i = 0; i < z; i++)
{for (j = i + 1; j < z; j++)
{if (!strcmp(pr[i].r, pr[j].r)) pr[j].l = '\0';}}
printf("\nOptimized Code\n");
for (i = 0; i < z; i++)
if (pr[i].l) printf("%c=%s\n", pr[i].l, pr[i].r);
return 0;}
```

OUTPUT:

```
Enter 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
b=c+d
e=c+d
f=b+e
r=f

Optimized Code
b=c+d
f=b+e
r=f
```