

1)Write a program to identify octal or hexadecimal using Lex

```
%{
%}
Oct [o][0-9]+
Hex [o][x|X][0-9A-F]+
%%
{Hex} printf("this is a hexadecimal number");
{Oct} printf("this is an octal number");
%%
main()
{
yylex();
}
int yywrap()
{
return 1;
}
Expected Output: ./a.outo5
this is an octal number
ox23
this is a hexadecimal number
```

2)Write a program to capitalize the comment using Lex

```
%{
#include<stdio.h>
#include<ctype.h>
int k;
void display(char *);
%}
letter [a-z]
com [//]
%%
{com} {k=1;}
{letter} {if(k==1) display(yytext);}
%%
main()
{
yylex();
}
void display(char *s)
{
int i;
for(i=0;s[i]!='\0';i++)
printf("%c", toupper(s[i]));
}
int yywrap()
{
return 1;
}
Expected Output:
lex caplex.lcc
lex.yy.c
./a.out
//hello world HELLO WORLD
```

3) Write a program to find complete real precision using Lex

```
%{
%}
integer ([0-9]+)
float    ([0-9]+\.[0-9]+)|([+|-]?[0-9]+\.[0-9]*[e|E][+|-][0-9]*)
%%
{integer} printf("\n %s is an integer\n",yytext);
{float}    printf("\n %s is a floating number\n",yytext);
%%
main()
{
yylex();
}
int yywrap()
{
return 1;
}
Expected Output:
lex real.l gcc
lex.yy.c
./a.out 1234
1234is an integer
```

4) Write a program to classify tokens as words

```
%{
    int tokenCount =0;
%}
%%
[a-zA-Z]+ {printf("%d WORD\","%s"\n",++tokenCount,yytext);
}[0-9]+ {printf("%dNUMBER\","%s"\n",++tokenCount,yytext);
}
[^a-zA-Z0-9]+ {printf("%dOTHER\","%s"\n", ++tokenCount,yytext); }
%%
main()
{
    yylex();
}
int yywrap()
{
    return 1;
}
```

Expected Output:

Input:

Hello! World ...this is 21 st century

OUTPUT:

1.WORD Hello

2.OTHER !

3.WORD World

4.OTHER ...

5.WORD this

6.WORD is

7.NUMBER 21

8.WORD st century

5) Write a Lex program to implement standalone scanner

```
%{
    int COMMENT=0;
}%
id [a-z][a-z0-9]*
%%
#.*      {printf("\n%s is a PREPROCESSOR
DIRECTIVE",yytext);}int|double|char {printf("\n\t%s is a
KEYWORD",yytext);}
if|then|endif    {printf("\n\t%s is a KEYWORD",yytext);}
else            {printf("\n\t%s is a KEYWORD",yytext);}
"/*"          {COMMENT=1;}
"*/"          {COMMENT=0;}
{id}\(         {if(!COMMENT)printf("\n\nFUNCTION\n\t%s",yytext);}
{id}(\[[0-9]*\])? {if(!COMMENT) printf("\n\tidentifier\t%s",yytext);}
\{            {if(!COMMENT) printf("\n BLOCK BEGINS");ECHO; }
\}            {if(!COMMENT)printf("\n BLOCK ends");ECHO; }
\'.*\'        {if(!COMMENT)printf("\n\t %s is a STRING",yytext);}
[+|-]?[0-9]+  {if(!COMMENT)printf("\n\t%s is a
NUMBER",yytext);}
\(            {if(!COMMENT)printf("\n\t");ECHO;printf("\t delim
openparanthesis\n");}
\)            {if(!COMMENT)printf("\n\t");ECHO;printf("\t delim closed
paranthesis");}
\;            {if(!COMMENT)printf("\n\t");ECHO;printf("\t delim semicolon");}
\=            {if(!COMMENT)printf("\n\t%s is an ASSIGNMENT
OPERATOR",yytext);}
\<|\>        {printf("\n\t %s is relational
operator",yytext);}+"|'|"-|'*'|'/' {printf("\n %s is an
operator\n",yytext);}
"\n" ;
%%
main(int argc ,char **argv)
{
    if (argc > 1)
        yyin = fopen(argv[1],"r");
    else
        yyin = stdin;
    yylex ();
    printf ("\n");
}
int yywrap()
{
    return 0;
}
```

Expected Output:

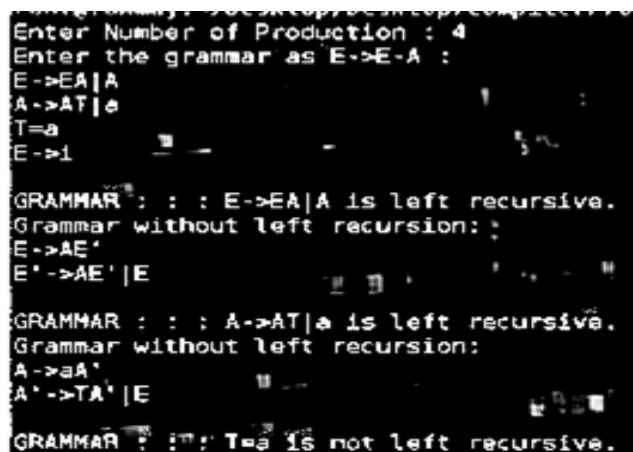
1. Save the file with .l extension\
2. Create a text file for eg: input.txt and write #include<stdio.h> , int
lex lexscanner.lcc
lex.yy.c
./a.out input.txt

#include<stdio.h> is a PREPROCESSOR DIRECTIVE
int is a KEYWORD

6) Write a C/C++ program to remove left recursion

```
#include<stdio.h>
#include<string.h>
#define SIZE 10
int main () {
char non_terminal;
char beta,alpha;
int num;
char production[10][SIZE];
int index=3; 10:
printf("Enter Number of Production : ");
scanf("%d",&num);
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("\nGRAMMAR : : : %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]!='|')
index++;
if(production[i][index]!=0) {
beta=production[i][index+1];
printf("Grammar without left recursion:\n");
printf("%c->%c%c%c\',"non_terminal,beta,non_terminal);
printf("\n%c\''-
>%c%c%c\''|E\n",non_terminal,alpha,non_terminal);
}
else
printf(" can't be reduced\n");
}
else
printf(" is not left recursive.\n");
index=3;
}
}
```

Expected Output:



```
Enter Number of Production : 4
Enter the grammar as E->E-A :
E->EA|A
A->AT|a
T=a
E->1

GRAMMAR : : : E->EA|A is left recursive.
Grammar without left recursion:
E->AE'
E'->AE'|E

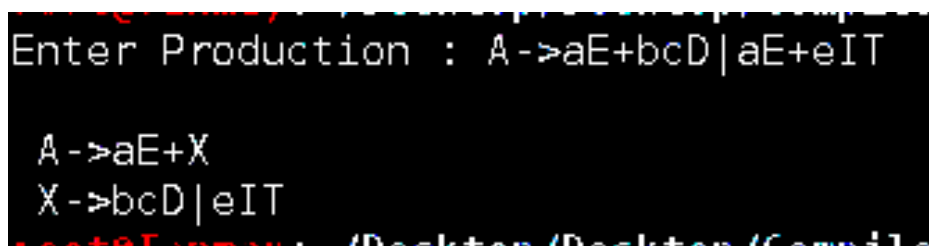
GRAMMAR : : : A->AT|a is left recursive.
Grammar without left recursion:
A->aA'
A'->TA'|E

GRAMMAR : : : T=a is not left recursive.
```

7) Write a C/C++ program to eliminate left factoring

```
#include<stdio.h>
#include<string.h>
int main()
{
    char gram[20],part1[20],part2[20],modifiedGram[20],newGram[20],tempGram[20];
    int i,j=0,k=0,l=0,pos;
    printf("Enter Production : A->");
    gets(gram);
    for(i=0;gram[i]!='|';i++,j++)
        part1[j]=gram[i];
    part1[j]='\0';
    for(j=++i,i=0;gram[j]!='\0';j++,i++)
        part2[i]=gram[j];
    part2[i]='\0';
    for(i=0;i<strlen(part1)||i<strlen(part2);i++)
    {
        if(part1[i]==part2[i])
        {
            modifiedGram[k]=part1[i];
            k++;
            pos=i+1;
        }
    }
    for(i=pos,j=0;part1[i]!='\0';i++,j++){
        newGram[j]=part1[i];
    }
    newGram[j++]='|';
    for(i=pos;part2[i]!='\0';i++,j++){
        newGram[j]=part2[i];
    }
    modifiedGram[k]='X';
    modifiedGram[++k]='\0';
    newGram[j]='\0';
    printf("\n A->%s",modifiedGram);
    printf("\n X->%s\n",newGram);
}
```

Expected Output:



```
Enter Production : A->aE+bcD|aE+eIT
A->aE+X
X->bcD|eIT
```

8) Write a program to implement recursive descent parser

```
#include<stdio.h>
#include<string.h>
void E(),E1(),T(),T1(),F();
int ip=0;
static char s[10];
int main()
{char k;
int l;
ip=0;
printf("enter the input");
scanf("%s",s);
printf("the string is :%s",s);E();
if(s[ip]=='$')
printf("\n string is accepted the length of string is %d",strlen(s)-1);
else
printf("\n string not accepted\n");
return 0;}
void E()
{
T();
E1();
return;
}
void E1()
{
if(s[ip]=='+')
{ip++;T();
E1();
}return;
}
void T()
{
F();
T1();
return;
}
void T1()
{
if(s[ip]=='*')
{ip++;F();
T1();
}return;}
void F()
{
if(s[ip]=='(')
{ip++;E();
if(s[ip]==')')
ip++;
}
else
if(s[ip]=='i')
ip++;
else
printf("\n id expected");
return;}
```

Expected Output:

```
cc recurparser.c
./a.out
enter the input
(i+i)*(i*i)$
the string is :(i+i)*(i*i)$
string is accepted the length of string is 11
```

9) Write a program to implement recursive descent parser

```
#include<stdio.h>
#include<iostream>
#include<string.h>
using namespace std;
char prol[7][10]={"S","A","A","B","B","C","C"};
char pror[7][10]={"A","Bb","Cd","aB","@","Cc","@"};
char prod[7][10]={"S->A","A->Bb","A->Cd","B->aB","B->@","C->Cc","C->@"}; char
first[7][10]={"abcd","ab","cd","a@","@","c@","@"}; char
follow[7][10]={"$","$","$","a$","b$","c$","d$"};
char table[5][6][10];
int numr(char c)
{
    switch(c)
    {
        case 'S': return 0;
        case 'A': return 1;
        case 'B': return 2;
        case 'C': return 3;
        case 'a': return 0;
        case 'b': return 1;
        case 'c': return 2;
        case 'd': return 3;
        case '$': return 4;
    }
    return(2);
}
int main(int argc, char *argv[])
{
    int i,j,k;
    for(i=0;i<5;i++)
        for(j=0;j<6;j++)
            strcpy(table[i][j]," ");
    printf("\nThe following is the predictive parsing table for the following grammar:\n");
    for(i=0;i<7;i++)
        printf("%s\n",prod[i]);
    printf("\nPredictive parsing table
is\n");fflush(stdin);
    for(i=0;i<7;i++)
    {
        k=strlen(first[i]);
        for(j=0;j<10;j++)
            if(first[i][j]!='@')
                strcpy(table[numr(prol[i][0])+1][numr(first[i][j])+1],prod[i]);
    }
    for(i=0;i<7;i++)
    {
        if(strlen(pror[i])==1)
        {
            if(pror[i][0]=='@')
            {
                k=strlen(follow[i]);
                for(j=0;j<k;j++)
                    strcpy(table[numr(prol[i][0])+1][numr(follow[i][j])+1],prod[i]);
            }
        }
    }
}
```

```

    }
}
strcpy(table[0][0]," ");
strcpy(table[0][1],"a");
strcpy(table[0][2],"b");
strcpy(table[0][3],"c");
strcpy(table[0][4],"d");
strcpy(table[0][5],"$");
strcpy(table[1][0],"S");
strcpy(table[2][0],"A");
strcpy(table[3][0],"B");
strcpy(table[4][0],"C");
printf("\n.....\n");
for(i=0;i<5;i++)
    for(j=0;j<6;j++)
    {
        printf("%-10s",table[i][j]);
        if(j==5)
            printf("\n.....\n");
    }
system("PAUSE"); // statement in Bloodshed dev c++ IDE requirement
}

```

Expected Output:

The following is the predictive parsing table for the following grammar:

$S \rightarrow A$
 $A \rightarrow Bb$
 $A \rightarrow Cd$
 $B \rightarrow aB$
 $B \rightarrow \epsilon$
 $C \rightarrow Cc$
 $C \rightarrow \epsilon$

Predictive parsing table is

	a	b	c	d	\$
S	$S \rightarrow A$	$S \rightarrow A$	$S \rightarrow A$	$S \rightarrow A$	
A	$A \rightarrow Bb$	$A \rightarrow Bb$	$A \rightarrow Cd$	$A \rightarrow Cd$	
B	$B \rightarrow aB$	$B \rightarrow \epsilon$	$B \rightarrow \epsilon$		$B \rightarrow \epsilon$
C			$C \rightarrow \epsilon$	$C \rightarrow \epsilon$	$C \rightarrow \epsilon$

Press any key to continue . . . _

10) Write a C/C++ program for LR parser table generation

```
#include<stdio.h>
#include<iostream>
using namespace std;
char stack[30];
int top=-1;
void push(char c)
{top++;
stack[top]=c;}
char pop()
{char c;
if(top!=-1)
{c=stack[top];
top--;
return c;}
return'x';}
void printstat()
{int i;
printf("\n\t\t $");
for(i=0;i<=top;i++)
printf("%c",stack[i]);}
int main(int argc, char *argv[])
{int i,j,k,l;
char s1[20],s2[20],ch1,ch2,ch3;
printf("\n\n\t\t LR PARSING");
printf("\n\t\t ENTER THE
EXPRESSION");scanf("%s",s1);
l=strlen(s1);
j=0;
printf("\n\t\t $");
for(i=0;i<l;i++)
{if(s1[i]=='i' && s1[i+1]=='d')
{s1[i]=' ';
s1[i+1]='E';
printstat();
printf("id");push('E');
printstat();}
else if(s1[i]=='+'||s1[i]=='-'||s1[i]=='*'||s1[i]=='/' ||s1[i]=='d')
{push(s1[i]);printstat();
}}
printstat(); l=strlen(s2);
while(l)
{ch1=pop(); if(ch1=='x')
{printf("\n\t\t $");break;
}if(ch1=='+'||ch1=='/'||ch1=='*'||ch1=='-')
{ch3=pop();
if(ch3!='E'){
printf("error");
exit(0);}
else{
push('E');
printstat();}
}ch2=ch1;
}
system("PAUSE");}
```

```
LR PARSING
ENTER THE EXPRESSIONid+id*id-id
$
$ id
$ E
$ E+
$ E+id
$ E+E
$ E+E*
$ E+E*id
$ E+E*E
$ E+E*E-
$ E+E*E-id
$ E+E*E-E
$ E+E*E-E
$ E+E*E
$ E+E
$ E
$ Press any key to continue . . . _
```

11) Write a program to implement parser using YACC

FILE 1: parser.l

```
%{
    #include "y.tab.h"
    extern int yylval;
}%

%%

[0-9]+ {yylval=atoi(yytext); return NUM;}
[\t]
\n return 0;
return yytext[0];

%%
int yywrap()
{
    return 0;
}
```

FILE 2 :parser.y

```
%token NUM
%%

cmd  :E {printf("%d\n", $1);}
      ;
E    :E '+' T {$$=$1+$3;}
      |T {$$=$1;}
      ;
T    :T '*' F {$$=$1*$3;}
      |F {$$=$1;}
      ;
F    : '(' E ')' {$$=$2;}
      |NUM {$$=$1;}
      ;

%%

int main()
{
    yyparse();
}
yyerror(char *s)
{
    printf("%s", s);
}
```

Expected Output:

```
lex parser.l yacc
-d parser.y
gcc lex.yy.c y.tab.c -ll -ly
./a.out
2+3
5
```

12) Write a program to implement a calculator using YACC

FILE 1 : cal.l

```
%{
#include<stdio.h>
#include "y.tab.h"
%}
%%
[0-9]+ {yylval.dval=atoi(yytext); return DIGIT;}
\n|.    return yytext[0];
%%
```

FILE 2 :Cal.y

```
%{
/* */
%}
%union
{
int dval;
}
%token <dval> DIGIT
%type <dval> expr
%type <dval> expr1

%%

line : expr '\n' {printf("%d\n", $1);}
    ;
expr : expr '+' expr1 {$$=$1+$3;}
    | expr '-' expr1 {$$=$1-$3;}
    | expr '*' expr1 {$$=$1*$3;}
    | expr '/' expr1 {$$=$1/$3;}
    | expr1
    ;
expr1: '(' expr ')' {$$=$2;}
    | DIGIT
    ;

%%

int main()
{
    yyparse();
}
yyerror(char *s)
{
    printf("%s",s);
}
```

Expected Output:

```
$ lex cal.l
$ yacc -d cal.y
$ gcc lex.yy.c y.tab.c -ll
$ ./a.out
1+2
3
```

13) Write a C/C++ program for intermediate code generation

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int i=1,j=0,no=0,tmpch=90;
char str[100],left[15],right[15];
void findopr();
void explore();
void fleft(int);
void fright(int);
struct exp
{
    int pos;
    char op;
}k[15];
int main()
{
    printf("\t\tINTERMEDIATE CODE GENERATION\n\n");
    printf("Enter the Expression :");
    scanf("%s",str);
    printf("The intermediate code:\n");
    findopr();
    explore(); }
void findopr()
{
    for(i=0;str[i]!='\0';i++)
    if(str[i]==':')
    {
        k[j].pos=i;
        k[j++].op=':';
    }
    for(i=0;str[i]!='\0';i++)
    if(str[i]=='/')
    {
        k[j].pos=i;
        k[j++].op='/';
    }
    for(i=0;str[i]!='\0';i++)
    if(str[i]=='*')
    {
        k[j].pos=i;
        k[j++].op='*';
    }
    for(i=0;str[i]!='\0';i++)
    if(str[i]=='+')
    {
        k[j].pos=i;
        k[j++].op='+';
    }
    for(i=0;str[i]!='\0';i++)
    if(str[i]=='-')
    {
        k[j].pos=i;
        k[j++].op='-';
    }
}
```

```

        return ;
    }
    void explore()
    {
        i=1;
        while(k[i].op!='\0')
        {
            fleft(k[i].pos);
            fright(k[i].pos);
            str[k[i].pos]=tmpch--;
            printf("\t%c := %s%c%s\t\t",str[k[i].pos],left,k[i].op,right);
            printf("\n");
            i++;
        }
        fright(-1);
        if(no==0)
        {
            fleft(strlen(str));
            printf("\t%s := %s",right,left);
            getch();
            exit(0);}
        printf("\t%s := %c",right,str[k[--i].pos]);
        getch();
    }
    void fleft(int x)
    {
        int w=0,flag=0;
        x--;
        while(x!= -1 &&str[x]!='+' &&str[x]!='*' &&str[x]!='=' &&str[x]!='\0' &&str[x]!='-'
        '&&str[x]!='/' &&str[x]!=':')
        {
            if(str[x]!='$' && flag==0)
            {
                left[w++]=str[x];
                left[w]='\0';
                str[x]='$';
                flag=1;
            }
            x--;
        }
    }
    void fright(int x)
    {
        int w=0,flag=0;
        x++;
        while(x!= -1 && str[x]!='+' &&str[x]!='*' &&str[x]!='\0' &&str[x]!='=' &&str[x]!='-'
        '&&str[x]!='/')
        {
            if(str[x]!='$' && flag==0)
            {
                right[w++]=str[x];
                right[w]='\0';
                str[x]='$';
                flag=1;
            }
            x++;
        }
    }
}

```

INTERMEDIATE CODE GENERATION

Enter the Expression :w:=a*b+c/d-e/f+g*h
The intermediate code:

```

Z := c/d
Y := e/f
X := a*b
W := g*h
U := X+Z
U := Y+W
T := U-U
w := T

```

14)write a Program to generate target machine code

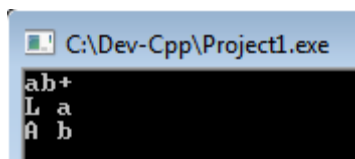
```
#include<stdio.h>
char stk[100],stktop=-1,cnt=0;
void push(char pchar)
{
stk[++stktop]=pchar;
}
char pop()
{
return stk[stktop--];
}
char checkoperation(char char1)
{
char oper;
if(char1=='+')
oper='A';
else if(char1=='-')
oper='S';
else if(char1=='*')
oper='M';
else if(char1=='/')
oper='D';
else if(char1=='@')
oper='N';
return oper;
}
int checknstore(char check)
{
int ret;
if(check!='+' && check!='-' && check!='*' && check!='/' && check!='@')
{
push(++cnt);
if(stktop>0)
printf("ST $%d\n",cnt);
ret=1;
}
else
ret=0;
return ret;
}
int main(int argc, char *argv[])
{
char msg[100],op1,op2,operation;
int i,val;
while(scanf("%s",msg)!=EOF)
{
cnt=0;
stktop=-1;
for(i=0;msg[i]!='\0';i++)
{
if((msg[i]>='A' && msg[i]<='Z') ||(msg[i]>='a' && msg[i]<='z'))
push(msg[i]);
else
{
op1=pop();
```

```

op2=pop();
printf("L %c\n",op2);
operation=checkoperation(msg[i]);
printf("%c %c\n",operation,op1);
val=checknstore(msg[i+1]);
while(val==0)
{
op1=pop();
cnt--;
operation=checkoperation(msg[++i]);
if(operation=='S' && stktop>=-1)
{
printf("N\n");
operation='A';
}
printf("%c %c\n",operation,op1);
val=checknstore(msg[i+1]);
}
}
}
}
system("PAUSE");
}

```

Output



15) Write a C/C++ program for code optimization

```
#include<stdio.h>
#include<conio.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;
    clrscr();
    printf("Enter the Number of
    Values:");scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("left: ");
        op[i].l=getche();
        printf("\tright: ");
        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",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';
            strcpy(pr[i].r,'\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);
    }
}
getch();
}

```

Expected Output:

Enter the Number of Values: 5

Left: a right: 9

Left: b 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

Eliminate Common Expression

b=c+d

b=c+d

f=b+b

r=:f

Optimized Code

b=c+d

f=b+b

r=:f

16) Write a C/C++ program for finding the FIRST for given grammar

```
#include<stdio.h>
#include<ctype.h>
int main()
{
    int i,n,j,k;
    char str[10][10],f;
    printf("Enter the number of productions\n");
    scanf("%d",&n);
    printf("Enter
grammar\n");
    for(i=0;i<n;i++)
    scanf("%s",&str[i]);
    for(i=0;i<n;i++)
    {
        f= str[i][0];
        int temp=i;
        if(isupper(str[i][3]))
        {
            repeat:
            for(k=0;k<n;k++)
            {
                if(str[k][0]==str[i][3])
                {
                    if(isupper(str[k][3]))
                    {
                        i=k;
                        goto repeat;
                    }
                    else
                    {
                        Printf("First(%c)=%c\n",f,str[k][3]);
                    }
                }
                else
                {
                    printf("First(%c)=%c\n",f,str[k][3]);
                }
            }
        }
    }
    else{
        C
    }
    i=temp;
}}
```

Expected Output:

cc first.c

./a.out

Enter the number of productions

3

Enter grammar

S->AB

A->aB-

>b

First(S)=a

First(A)=a

First(B)=b

17) Write a C/C++ program for finding the FOLLOW for given grammar

```
#include<stdio.h>
main()
{
    int np,i,j,k;
    char prods[10][10],follow[10][10],Imad[10][10];
    printf("enter no. of productions\n");
    scanf("%d",&np);
    printf("enter grammar\n");
    for(i=0;i<np;i++)
    {
        scanf("%s",&prods[i]);
    }

    for(i=0; i<np; i++)
    {
        if(i==0)
        {
            printf("Follow(%c) = $\n",prods[0][0]);
        }
        for(j=3;prods[i][j]!='\0';j++)
        {
            int temp2=j;
            if(prods[i][j] >= 'A' && prods[i][j] <= 'Z')
            {
                if((strlen(prods[i])-1)==j)
                {
                    printf("Follow(%c)=Follow(%c)\n",prods[i][j],prods[i][0]);
                }
                int temp=i;
                char f=prods[i][j];
                if(!isupper(prods[i][j+1])&&(prods[i][j+1]!='\0'))
                printf("Follow(%c)=%c\n",f,prods[i][j+1]);
                if(isupper(prods[i][j+1]))
                {
                    repeat:
                    for(k=0;k<np;k++)
                    {
                        if(prods[k][0]==prods[i][j+1])
                        {
                            if(!isupper(prods[k][3]))
                            {
                                printf("Follow(%c)=%c\n",f,prods[k][3]);
                            }
                            else
                            {
                                i=k;
                                j=2;
                                goto repeat;
                            }
                        }
                    }
                }
                i=temp;
            }
            j=temp2;}}}
    j=temp2;}}}
```

Expected Output:

./a.out

Enter no. of productions: 3

Enter grammar

S->AB

A->a

B->b

Follow(S)=\$

Follow(A)=b

Follow(B)=Follow(S)