SRI KRISHNA COLLEGE OF ENGINEERING AND TECHNOLOGY
KUNIAMUTHUR, COIMBATORE - 641 008

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(AN AUTONOMOUS INSTITUTION,
AFFILIATED TO ANNA UNIVERSITY, CHENNAI)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

LAB MANUAL

Laboratory Name	**: Compiler Design Laboratory**
Course Code	**: 20CS604**
Degree & Branch	**: B.E – Computer Science and Engineering**
Semester	**: VI**
Academic year	**: 2022-2023**
Batch	**: 2020-2024**
Class	**: III B.E CSE**

LIST OF EXPERIMENTS

1. Implementation of lexical analyzer using C and LEX TOOL.
2. Implementation of a calculator that takes an expression (with digits, + and *), computes and prints its value, using YACC.
3. Implementation of a parser using LEX and YACC.
4. Implementation of symbol table
5. Implementation of Predictive parsing.
6. Implementation of Shift Reduce Parsing Algorithm.
7. Implementation of LR parsing.
8. Implementation of front end of a compiler that generates the three address code for a simple language with One data type integer, arithmetic operators, relational operators, variable declaration statement, one conditional construct, one iterative construct and assignment statement.
9. Implementation of back end of the compiler which takes the three address code as input and produces assembly language instructions that can be assembled and run using an 8086 assembler. The target assembly instructions can be simple move, add, sub, and jump.
10. Implementation of the code optimizer phase of a compiler that eliminates dead code and common sub-expressions.

Ex: No: 1a	Implementation of lexical analyzer using C programming
Date:	

Aim

To implement lexical analyzer using C programming.

Program

```
#include<string.h>
#include<ctype.h>
#include<stdio.h>
void keyword(char str[10])
{
   char keywords[10][10]={"int","float","char","while","do","for","if"};
   if(!strcmp(*keywords,str))
   {
     printf("\n%s is a keyword",str);
   }
    else
   printf("\n%s is an identifier",str);

}
void main()
{
   FILE *f1,*f2,*f3,*f4;
   char c,str[10],st1[10];

   int num[100],tokenvalue=0,i=0,j=0,k=0;
   printf("\nEnter the c program\n");
   f1=fopen("input","w");
   while((c=getchar())!=EOF)
   putc(c,f1);
   fclose(f1);
   f1=fopen("input","r");
   f2=fopen("identifier","w");
   f3=fopen("specialchar","w");
   f4=fopen("operators","w");
   while((c=getc(f1))!=EOF)
   {
      if(isdigit(c))
      {
         tokenvalue=c-'0';
         c=getc(f1);
```

```
        while(isdigit(c))
        {
            tokenvalue*=10+c-'0';
            c=getc(f1);
        }
        num[i++]=tokenvalue;
        ungetc(c,f1);
    }
    else if(isalpha(c))
    {
        putc(c,f2);
        c=getc(f1);
        while(isdigit(c)||isalpha(c)||c=='_'||c=='$')
        {
            putc(c,f2);
            c=getc(f1);
        }
        putc(' ',f2);
        ungetc(c,f1);
    }

    else if(c=='+' || c=='-' ||c=='*' ||c=='<'||c=='>'||c=='/'||c=='&'||c=='%' ||c=='^' ||c== '=')
    putc(c,f4);
    else
    putc(c,f3);
}
fclose(f4);
fclose(f2);
fclose(f3);
fclose(f1);
printf("\nThe constants are ");
for(j=0;j<i;j++)
printf("%d",num[j]);
printf("\n");
f2=fopen("identifier","r");
k=0;
printf("The keywords and identifiers are:");
while((c=getc(f2))!=EOF)
{
    if(c!=' ')
    str[k++]=c;
    else
    {
        str[k]='\0';
        keyword(str);
        k=0;
    }
}
```

```
    fclose(f2);
    f3=fopen("specialchar","r");
    printf("\nSpecial characters are ");
    while((c=getc(f3))!=EOF)
    printf("%c ",c);
    fclose(f3);
    f4=fopen("operators","r");
    printf("Operators are ");
    while((c=getc(f4))!=EOF)
    printf("%c ",c);
    printf("\n");
    fclose(f4);
}
```

Output:

$ cc lex_test.c
$ ./a.out

Enter the c program
a=b+6;

The constants are 6
The keywords and identifiers are:
a is an identifier
b is an identifier
Special characters are ;
Operators are =+

Viva Questions:

1. Describe the Analysis-Synthesis Model of compilation.
2. Define a symbol table.
3. Define tokens, patterns and lexemes.
4. What are the possible error recovery actions in lexical Analyzer?
5. What are the three general approaches to the implementation of a Lexical Analyzer?

Ex: No:1b	Implementation of Lexical analyzer using LEX
Date:	

Aim:

To implement the lexical analyzer using LEX tool.

Program for Count the word, character, space and line from the file using LEX:

```
%{
int c=0,w=0,l=0,s=0;
%}

%%
[\n] l++;s++;
[\t "] s++;
[^ " \t \n]+ w++;c+=yyleng;
%%
int main(int argc,char *argv[])
{
	if(argc==2)
	{
		yyin=fopen(argv[1],"r");
		yylex();
		printf("\nNumber of spaces = %d",s);
		printf("\nNumber of characters = %d",c);
		printf("\nNumber of lines = %d",l);
		printf("\nNumber of words = %d\n",w);
	}
	else
		printf("ERROR");
}
```

abc.txt

```
hai
hello
how are you?
```

Output :

$ lex l1.l
$ cc lex.yy.c -ll
$ ./a.out abc.txt

```
Number of spaces = 5
Number of characters = 18
Number of lines = 3
```

Number of words = 5

PROGRAM for Lexical Analyzer using LEX tool :

```
%{
#include<stdio.h>
%}
letter [a-zA-Z]
digit[0-9]
operators [+*/=%&|<>-]
specialcharacters [();{}"]
%%
(#include<stdio.h>|void|main|int|float|char|printf|while|do|for|if|else|double|break|continue|scanf|switch|case)+ {printf(" Keyword ");}
{letter}({letter}|{digit})* {printf(" Variable ");}
{digit}+ {printf(" Number ");}
{operators}+ {printf(" Operator ");}
{specialcharacters}+ {printf(" Specialcharacter ");}
%%
int main(int argc,char *argv[])
{
      yyin = fopen(argv[1],"r");
      yylex();
}
```

abc.c

```
#include<stdio.h>
void main()
{
a=b+3;
printf("Hello world");
}
```

Output :

$ lex l2.l
$ cc lex.yy.c -ll
$ ./a.out abc.c

```
 Keyword
Keyword  Keyword  Special character
Special character
Variable  Operator  Variable  Operator  Number  Special character
Keyword  Special character  Variable  Variable  Special character
```

Viva Questions:

1. Implementation steps for LEX tool.
2. What is lookahead operator.
3. Define NFA.
4. Write the pattern for the 0(0|1)*0 and (0|1)*0(0|1)(0|1)
5. Examples of Lex regular expression.

Ex: No:2	Implementation of a calculator that takes an expression (with digits, + and *), computes and prints its value, using YACC.
Date:	

Aim:

To implement calculator that takes an expression (with digits, + and *), computes and prints its value, using YACC.

Program 1 :

//calc.l

```
%{
/* Definition section */
#include<stdio.h>
#include "y.tab.h"
extern int yylval;
%}

/* Rule Section */
%%
[0-9]+ {
                yylval=atoi(yytext);
                return NUMBER;

        }
[\t] ;

[\n] return 0;

. return yytext[0];

%%

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

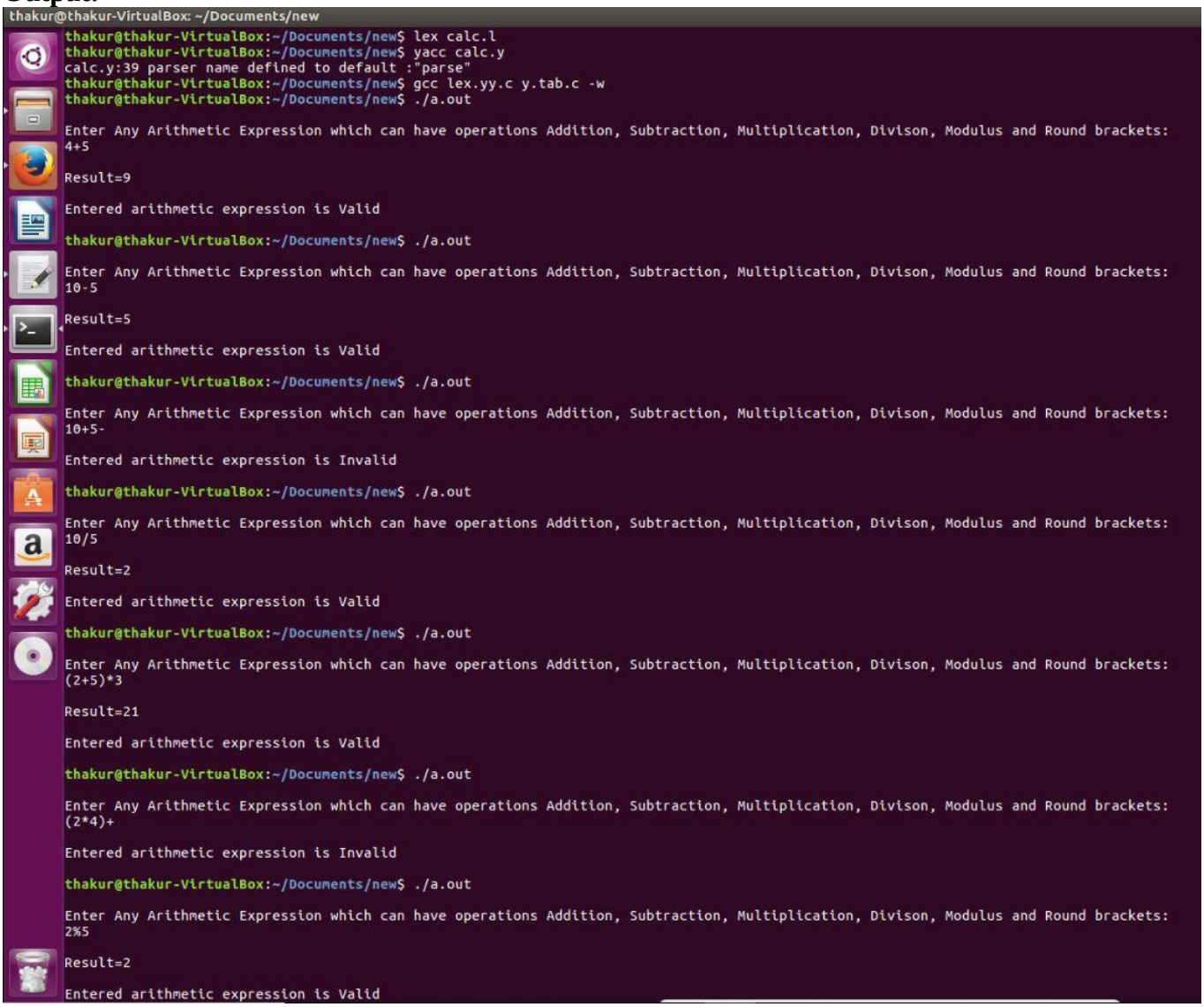
//calc.y

```
%{
/* Definition section */
#include<stdio.h>
int flag=0;
```

```
%}

%token NUMBER

%left '+' '-'

%left '*' '/' '%'

%left '(' ')'

/* Rule Section */
%%

ArithmeticExpression: E{

                        printf("\nResult=%d\n", $$);

                        return 0;

                        };
E:E'+'E {$$=$1+$3;}

|E'-'E {$$=$1-$3;}

|E'*'E {$$=$1*$3;}

|E'/'E {$$=$1/$3;}

|E'%'E {$$=$1%$3;}

|'('E')' {$$=$2;}

| NUMBER {$$=$1;}

;

%%

//driver code
void main()
{
printf("\nEnter Any Arithmetic Expression which
                                        can have operations Addition,
                                        Subtraction, Multiplication, Division,
                                                        Modulus and Round brackets:\n");

yyparse();
if(flag==0)
```

```
printf("\nEntered arithmetic expression is Valid\n\n");
}

void yyerror()
{
printf("\nEntered arithmetic expression is Invalid\n\n");
flag=1;
}
```

Output:



Ex: No:3	Implementation of Parser using LEX and YACC tool
Date:	

Aim:

To implement YACC using C programming.

Program 1 :

//lexx1.l

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

%%
[0-9]+ {yylval=atoi(yytext);
printf("Scanned the number %d\n",yylval);
return NUMBER;}
[a-zA-Z]+ {printf("Scanned a name\n");
return NAME;}
[\t] {printf("Skipped whitespace\n");}
\n {return 0;}
{printf("Found other data \"%s\"\n",yytext);
return yytext[0];
}
%%
```

//yacc1.y

```
%{
#include<stdio.h>
%}

%token NAME NUMBER

%%
stmt:S {printf("SUCCESS\n");}
S : '('L')'{}
   |NAME{}
   |NUMBER{}
   ;
L : L','S{}
   |S{}
```

```
   ;
%%

int main(void)
{
  return yyparse();
}
int yyerror(char *msg)
{
   return fprintf(stderr,"YACC:%s\n",msg);
}
```

Output:

```
$ lex lexx1.l
$ yacc -d yacc1.y
$ cc lex.yy.c y.tab.c -ll
$ ./a.out
(id,(id,1))
Found other data "("
Scanned a name
Found other data ","
Found other data "("
Scanned a name
Found other data ","
Scanned the number 1
Found other data ")"
Found other data ")"
SUCCESS

$ ./a.out

(id))
Found other data "("
Scanned a name
Found other data ")"
SUCCESS
Found other data ")"
YACC:syntax error

$ ./a.out
((id,id),(id))
Found other data "("
Found other data "("
Scanned a name
Found other data ","
Scanned a name
Found other data ")"
Found other data ","
```

```
Found other data "("
Scanned a name
Found other data ")"
Found other data ")"
SUCCESS
```

Program 2 :

//lexx1.l

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

%%
[0-9]+ {yylval=atoi(yytext);
printf("Scanned the number %d\n",yylval);
return NUMBER;}
[a-zA-Z]+ {printf("Scanned a name\n");
return NAME;}
[\t] {printf("Skipped whitespace\n");}
\n {return 0;}
. {printf("Found other data \"%s\"\n",yytext);
return yytext[0];
}
%%
```

//yacc2.y

```
%{
#include<stdio.h>
%}

%token NAME NUMBER

%%
stmt:E {printf("SUCCESS\n");}
E : E'+'T{}
   |T
   ;
T : T'*'F{}
   |F
   ;
F : '('E')'{}
   |NAME{}
   |NUMBER{}
   ;
```

```
%%
int main(void)
{
  return yyparse();
}
int yyerror(char *msg)
{
   return fprintf(stderr,"YACC:%s\n",msg);
}
```

OUTPUT :

```
$ lex lexx1.l
$ yacc -d yacc2.y
$ cc lex.yy.c y.tab.c -ll
$ ./a.out
id*id+id
Scanned a name
Found other data "*"
Scanned a name
Found other data "+"
Scanned a name
SUCCESS

$ ./a.out
id*
Scanned a name
Found other data "*"
YACC:syntax error
```

Viva Questions:

1. Define parser.
2. Define the CFG.
3. Given grammar find the first and follow.
4. How to implement YACC tool.
5. What is annotated parse tree?

Ex: No:4	Implementation of Symbol table
Date:	

Aim

To implement the symbol table using C programming.

Program:

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
struct symtab
{
  int lineno;
  char var[25],dt[25],val[10];
}sa[20];

void main()
{

   int i=0,j,k,max,f=0,xx,h,m,n,l,r,ty=1,m1,line=0;
   char s[25],typ[25],temp[25],gar[]="garbage",t[25],got[10],e[10];
   float m2;
   FILE *fn,*ft,*fp;

//sa[1].line=1;

fn=fopen("input.txt","r");
printf("\n\nSYMBOL TABLE MANAGEMENT\n\n");
printf("Variable\tDatatype\tLine.no.\t\tValue\n");
          while(!(feof(fn)))
          {
            fscanf(fn,"%s",s);
            if((strcmp(s,"int")==0)||(strcmp(s,"float")==0))
          {
              strcpy(typ,s); line++;
                    while(s,";"!=0)
                     {
                 i++;
                       max=i;      sa[i].lineno=line;
                       fscanf(fn,"%s",s);
                                    strcpy(sa[i].var,s);
                                    strcpy(sa[i].dt,typ);
```

```
                        fscanf(fn,"%s",s);
                        if(strcmp(s,"=")==0)
                        {
                        fscanf(fn,"%s",s);
                        strcpy(sa[i].val,s);
                        fscanf(fn,"%s",s);
                        }
                        else
                                strcpy(sa[i].val,gar);
                        if(strcmp(s,",")==0)
                                continue;
                        else break;
                }
            }
  else if(strcmp(s,"char")==0)
       {
        strcpy(typ,s); line++;
          while(strcmp(s,";")!=0)
          {
           i++;
            max=i; sa[i].lineno=line;
fscanf(fn,"%s",s);
strcpy(sa[i].var,s);
strcpy(sa[i].dt,typ);
fscanf(fn,"%s",s);
                if(strcmp(s,"=")==0)
                {
                fscanf(fn,"%s",s);
                fscanf(fn,"%s",s);
                strcpy(sa[i].val,s);
                fscanf(fn,"%s",s);
                fscanf(fn,"%s",s);
                }

        }//while
                fscanf(fn,"%s",s);
                        if(strcmp(s,",")==0)
                                continue;
                        }//else if
    }//while
for(i=1;i<=max;i++)
printf("\n%s\t\t%s\t\t%d\t\t%s\n",sa[i].var,sa[i].dt,sa[i].lineno,sa[i].val);
fclose(fn);
}
```

Input File:

```
int a , b = 5 ;
float c ;
char d = " a " ;
```

Output:

$ cc symbol.c
$ ./a.out

SYMBOL TABLE MANAGEMENT

Variable	Datatype	Line.no.	Value
a	int	1	garbage
b	int	1	5
c	float	2	garbage
d	char	3	a

Viva Questions:

1. Define a symbol table.
2. List the attributes of Symbol table for Simple C program.
3. List the methods for token specification.
4. Define regular language.
5. Brief RE pattern for tokens.
6. Why symbol table management is required?
7. Give the structure of the symbol table.
8. Which phases are connected with symbol table.
9. How symbol tables are accessed by each phase

Ex: No:5	Implementation of Predictive Parser
Date:	

Aim:

To implement Predictive parser using C Programming.

Program:

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
struct tran
{
	char node;
	int n,k,g,fi,fo;
	char t[5][20];
	char first[10],follow[10];
	char par[10][10];
}b[20];
void first(int);
void follow(int);
void get_parse_table();
void manipulate_string();
int count,c=0;
char a[10][20],t[15],stack[10],string[10],e;
int main()
{
	int i,j,k,h;
	printf("Enter the productions:\n");
	for(count=0;;count++)
	{
		scanf("%s",a[count]);
		if(!strcmp(a[count],"quit"))
			break;
		b[count].node=a[count][0];
		for(i=3,j=0;i<strlen(a[count]);i++)
		{
			if(a[count][i]=='/')
			{
				b[count].n++;
				j=0;
				i++;
			}
	 if(!isupper(a[count][i])&&a[count][i]!='@')
```

```
        {
                for(k=0,h=0;k<c;k++)
                        if(a[count][i]==t[k])
                    h=1;
          if(h==0)
                        t[c++]=a[count][i];
        }
                        b[count].t[b[count].n][j++]=a[count][i];
                }

        }
  t[c++]='$';
        for(i=0;i<count;i++)
                if(b[i].k==0)
                        first(i);
        b[0].follow[b[0].fo++]='$';
        for(i=0;i<count;i++)
        if(b[i].g==0)
                        follow(i);
  get_parse_table();
  printf("NT\t");
  for(i=0;i<c;i++)
        printf("%c\t",t[i]);
  printf("\n");
  for(i=0;i<50;i++)
  printf("-");
  printf("\n");
  for(i=0;i<count;i++)
  {
        printf("%c\t",b[i].node);
    for(j=0;j<c;j++)
    {
        if(b[i].par[j][0])
                printf("%c->%s ",b[i].node,b[i].par[j]);
        printf("\t");
    }
    printf("\n");
  }
  manipulate_string();
}
void first(int x)
{
        int i,h,j,k;
        b[x].k=1;
        for(i=0;i<=b[x].n;i++)
        {
                for(j=0,h=0;j<count;j++)
                {
```

```
                        if(b[x].t[i][0]==b[j].node)
                        {
                                if(b[j].k==0)
                                        first(j);
                                h=1;
                                for(k=0;k<b[j].fi;k++)
                                        b[x].first[b[x].fi++]=b[j].first[k];
                        }
                }
                if(h==0)
                        b[x].first[b[x].fi++]=b[x].t[i][0];
        }
}
void follow(int x)
{
        int i,j,l,h,n,k,g;
  b[x].g=1;
        for(i=0;i<count;i++)
        {
                for(j=0;j<=b[i].n;j++)
                {
                        for(k=strlen(b[i].t[j])-1;k>=0;k--)
                        {
                                if(b[x].node==b[i].t[j][k])
                                {
                                        if(k==strlen(b[i].t[j])-1)
                                        {
                if(b[i].g==0)
                                                        follow(i);
                                                for(l=0;l<b[i].fo;l++)
                                                        b[x].follow[b[x].fo++]=b[i].follow[l];
                                        }
                                        else
                                        {
                                                for(l=0,h=0;l<count;l++)
                                                {
                                                        if(b[l].node==b[i].t[j][k+1])
                                                        {
                                                                for(n=0;n<b[l].fi;n++)
                                                                {
                                                                        if(b[l].first[n]!='@')

        b[x].follow[b[x].fo++]=b[l].first[n];
                                                                        else
                                                                        {
                                                                                if(b[i].g==0)
                                                                                        follow(i);
```

```
        for(g=0;g<b[i].fo;g++)

        b[x].follow[b[x].fo++]=b[i].follow[g];
                                                                }
                                                        }
                                                        h=1;
                                                }
                                        }
                                        if(h==0)
                                                b[x].follow[b[x].fo++]=b[i].t[j][k+1];
                                }
                return;
                                }
                        }
                }
        }
}
void get_parse_table()
{
        int i,j,k,t1,l,n;
  char temp[5];
  for(i=0;i<count;i++)
  {
        for(j=0;j<=b[i].n;j++)
    {
        t1=0;
        if(b[i].t[j][0]=='@')
                for(k=0;k<b[i].fo;k++)
                        temp[t1++]=b[i].follow[k];
        else if(!isupper(b[i].t[j][0]))
                temp[t1++]=b[i].t[j][0];
         else
                for(k=0;k<b[i].fi;k++)
                        temp[t1++]=b[i].first[k];
         for(l=0;l<t1;l++)
           for(n=0;n<c;n++)
                if(t[n]==temp[l])
                  strcat(b[i].par[n],b[i].t[j]);
    }
  }
}
void manipulate_string()
{
        int top=0,i=0,h,h1,j,k,n;
        printf("\nEnter the string:");
        scanf("%s",string);
        string[strlen(string)]='$';
```

```
stack[top++]='$';
stack[top++]=b[0].node;
printf("\nSTACK\tSTRING");
while(1)
{
        printf("\n");
        for(k=0;k<top;k++)
        printf("%c",stack[k]);
        printf("\t");
        for(k=i;k<strlen(string);k++)
        printf("%c",string[k]);
        if(stack[top-1]==string[i])
        {
                if(string[i]=='$')
                {
                        printf("\n***String accepted***");
                        break;
                }
                top--;
                i++;
        }
        else if(isupper(stack[top-1])&&!isupper(string[i]))
        {
                for(j=0,h=0;j<count;j++)
                        if(b[j].node==stack[top-1])
                        {
                                h=1;
                                break;
                        }
for(k=0,h1=0;k<c;k++)
                        if(string[i]==t[k])
                        {
                                h1=1;
                                break;
                        }
if(h==0||h1==0)
{
                        printf("\n***Wrong Symbol***");
                        break;
}
                if(b[j].par[k][0])
                {
                                top--;
                                for(n=strlen(b[j].par[k])-1;n>=0;n--)
                                        if(b[j].par[k][n]!='@')
                                                stack[top++]=b[j].par[k][n];
                }
                else
```

```
                {
                        printf("\n**String not be accepted**");
                        break;
                }
        }
        else
        {
                printf("\n***String not be accepted***");
                break;
        }
    }
  printf("\nDO U WANT TO ENTER ANOTHER STRING(Y/N):");
  scanf(" %c",&e);
  if(e=='Y'||e=='y')
        manipulate_string();
}
```

Output :

$ cc predictive.c
$ ./a.out

```
Enter the productions:
E->TX
X->+TX/@
T->FY
Y->*FY/@
F->(E)/i
quit
NT      +       *       (       )       i       $
------------------------------------------------
E                       E->TX                   E->TX
X       X->+TX                          X->@            X->@
T                       T->FY                   T->FY
Y       Y->@    Y->*FY                  Y->@            Y->@
F                       F->(E)          F->i

Enter the string:i+i*i

STACK           STRING
$E      i+i*i$
$XT     i+i*i$
$XYF    i+i*i$
$XYi    i+i*i$
$XY     +i*i$
$X      +i*i$
```

```
$XT+  +i*i$
$XT   i*i$
$XYF  i*i$
$XYi  i*i$
$XY   *i$
$XYF*       *i$
$XYF  i$
$XYi  i$
$XY   $
$X    $
$     $
***String accepted***
DO U WANT TO ENTER ANOTHER STRING(Y/N):Y

Enter the string:i+*

STACK       STRING
$E    i+*$i$
$XT   i+*$i$
$XYF  i+*$i$
$XYi  i+*$i$
$XY   +*$i$
$X    +*$i$
$XT+  +*$i$
$XT   *$i$
**String not be accepted**
DO U WANT TO ENTER ANOTHER STRING(Y/N):N
```

Viva Questions:

1. What are the disadvantages of operator precedence parsing?
2. What is a predictive parser?
3. Eliminate left recursion from the following grammar A->Ac/Aad/bd/c.
4. What is LL (1) grammar? Give the properties of LL (1) grammar.
5. Give the algorithm for Left Factoring a Grammar.

Ex: No:6	Implementation of Shift Reduced Parser
Date:	

Aim:

To implement Shift Reduced Parser using C programming.

Program :

```
#include<stdio.h>
#include<string.h>
struct stack
{
   char s[20];
   int top;
};
struct stack st;
int isempty()
{   return (st.top==1);
}
void push(char p)
{
   st.s[st.top++]=p;
}
char pop()
{
   if(isempty())
      printf("stack empty");
   else
      return st.s[st.top--];
}
void disp()
{   int i;
   for(i=0;i<st.top;i++)
      printf("%c",st.s[i]);
}
int reduce(int *j,char rp[10][10],int n)
{   int i,t,k;
   char u[10];
   t=st.top-1;
   for (i=0;i<=st.top;i++)
   {   u[i]=st.s[t];
      u[i+1]='\0';
      for(k=0;k<n;k++)
      {
         if(strcmp(rp[k],u)==0)
```

```
                {
                   st.top=st.top-i-1;
                   return k;
            }
         }
         t--;
      }
      return 99;
}
int shift(char ip[],int *j)
{   push(ip[*j]);
      (*j)++;
      disp();
      return 1;
}
void main()
{   int n,i,j=0,k,h;
      char lp[10];
      char ip[10];
      char rp[10][10];
      st.top=0;
      printf("\nEnter the number of productions:");
      scanf("%d",&n);
      for(i=0;i<n;i++)
      {      printf("\nEnter the left side of the production %d:",i+1);
               scanf(" %c",&lp[i]);
               printf("\nEnter the right side of the production %d:",i+1);
               scanf("%s",rp[i]);
      }
      printf("\nEnter the input:");
      scanf("%s",ip);

printf("================================================================
==");
      printf("\nSTACK               INPUT                    OUTPUT        ");

printf("\n================================================================
===\n");
      strcat(ip,"$");
      push('$');
      printf("$                  %s         \n",ip);
      while(!(st.s[st.top-1]==lp[0]&&st.s[st.top-2]=='$'&&(j==(strlen(ip)-1))&&st.top==2))
      {
         if((h=reduce(&j,rp,n))!=99)
         {   push(lp[h]);disp();printf("\t\t\t");
            for(k=j;k<strlen(ip);k++)
               printf("%c",ip[k]);
            printf("\t\t\tReduce %c->%s\n",lp[h],rp[h]);
```

```
        }
        else if(shift(ip,&j))
        {     printf("\t\t\t");
                  for(k=j;k<strlen(ip);k++)
                 printf("%c",ip[k]);
              printf("\t\t\tshift %c\n",ip[j-1]);
        }
    }
    disp();
    printf("\t\t\t");
    for(k=j;k<strlen(ip);k++)
       printf("%c",ip[k]);
    printf("\t\t\taccept\n");
}
```

OUTPUT :

```
$ cc shift.c
$ ./a.out

Enter the number of productions:3

Enter the left side of the production 1:E

Enter the right side of the production 1:E+E

Enter the left side of the production 2:E

Enter the right side of the production 2:E*E

Enter the left side of the production 3:E

Enter the right side of the production 3:i

Enter the input:i+i*i
=============================================================
STACK            INPUT                OUTPUT
=============================================================
$                 i+i*i$
$i                   +i*i$              shift i
$E                   +i*i$              Reduce E->i
$E+                  i*i$               shift +
$E+i                 *i$                shift i
$E+E                 *i$                Reduce E->i
$E                   *i$                Reduce E->E+E
$E*                  i$                 shift *
$E*i                 $                  shift i
```

$E*E	$	Reduce E->i
$E	$	Reduce E->E*E
$E	$	accept

Viva Questions :

1. Define LR (0) items.
2. What do you mean by viable prefixes?
3. Define handle.
4. What is phrase level error recovery?
5. What are the disadvantages of operator precedence parsing?

Ex: No:7	Implementation of SLR Parser
Date:	

Aim:

To implement the SLR parser using C Programming.

Program:

```
#include<stdio.h>
#include<string.h>
struct table
{
        char op;
        char set;
        int value;
}ta[50][50];
void main()
{
    char nt[20],t[20],temp,ip[50],s[50],prod[20][20],rhs[20][20],lhs[20][20],temparr[20];
    int non,not,i,j,l,ni,ch=1,n,tos,n1,temp1,np,no;
    printf("Enter the no of non-terminals ");
    scanf("%d",&non);
    for ( i=0;i<non;i++ )
    {
            printf("Enter the non terminal ");
            scanf("%s",&nt[i]);
    }
    printf("Enter the no of terminals ");
    scanf("%d",&not);
    for ( i=non;i<not+non;i++ )
    {
            printf("Enter the terminal ");
            scanf("%s",&t[i-non]);
            nt[i]=t[i-non];
    }
    nt[i]='$';
    nt[i+1]='\0';
    printf("Enter the no of items ");
    scanf("%d",&ni);
    for ( i=0;i<ni;i++ )
    {
            printf("Enter the no of entries in item %d ",i);
            scanf("%d",&ch);
            for ( l=0;l<ch;l++ )
            {
                    printf("Enter the terminal/non-terminal ");
```

```
                scanf("%s",&temp);
                for ( j=0;temp!=nt[j];j++);
                printf("Enter the operation - S:Shift, R:Reduce ");
                scanf("%s",&ta[i][j].op);
                printf("Enter the step ");
                scanf("%d",&ta[i][j].value);
                ta[i][j].set='t';
        }
}
for ( i=0;i<=non+not+3;i++ )
   printf("%c\t",nt[i]);
printf("\n");
for ( i=0;i<ni;i++ )
{
        for ( j=0;j<=non+not+3;j++ )
        {
                if ( ta[i][j].set == 't' )
                        printf("%c%d\t",ta[i][j].op,ta[i][j].value);
                else
                        printf("\t");
        }
        printf("\n");
}
printf("Enter the no of productions ");
scanf("%d",&np);
for ( i=0;i<np;i++ )
{
        scanf("%s",prod[i]);
        lhs[i][0]=prod[i][0];
        lhs[i][1]='\0';
        for ( j=3;j<strlen(prod[i]);j++ )
                rhs[i][j-3]=prod[i][j];
        rhs[i][j-3]='\0';
}
ch=1;
while ( ch == 1 )
{
        printf("Enter the input string ");
        scanf("%s",ip);
        n=strlen(ip);
        ip[n]='$';
        ip[n+1]='\0';

        s[0]='$';
        s[1]='0';
        s[2]='\0';
        tos=1;
        i=0;
```

```
            n1=s[tos]-48;
            printf("Stack\tInput\n%s\t%s\n",s,ip);
            while (i<strlen(ip))
            {
                  if ( i != 0 )
                        1=s[tos];
                  for(l=0;nt[l]!=ip[i];l++);
                  temp=ta[n1][l].op;
                  if ( temp != 'S' && temp != 's' && temp !='r' && temp !='R' && temp !='-' )
                        break;
                  temp1=ta[n1][l].value;
                  if ( temp == 'S' || temp == 's' )
                  {
                     s[++tos]=ip[i++];
                        s[++tos]=temp1;
                        s[tos+1]='\0';
                  }
                  else
                  {
                        no=2*strlen(rhs[temp1-1]);
                        for ( j=0;j<no;j++ )
                              s[tos--]='\0';
                        s[++tos]=lhs[temp1-1][0];
                        if ( s[tos-1] == '0' )
                              n1=0;
                        else
                              n1=s[tos-1];
                        for ( l=0;nt[l]!=s[tos];l++);
                        if (ta[n1][l].value == 0 )
                              s[++tos]='0';
                        else
                              s[++tos]=ta[n1][l].value;
                  }
                  for ( l=0;l<strlen(s);l+=2)
                        printf("%c%d",s[l],s[l+1]==48?0:s[l+1]);
                  printf("\t\t");
                  for ( l=i;l<strlen(ip);l++ )
                        printf("%c",ip[l]);
                  printf("\n");
            }
            if ( s[tos] == 1 && ip[i] == '$' )
                  printf("String Accepted!");
            else
                  printf("String Not accepted.");
            printf("Do you want to enter another string?");
            scanf("%d",&ch);
      }
}
```

OUTPUT:
$ cc slr.c
$ ./a.out

```
Enter the no of non-terminals 3
Enter the non terminal E
Enter the non terminal T
Enter the non terminal F
Enter the no of terminals 5
Enter the terminal i
Enter the terminal +
Enter the terminal *
Enter the terminal (
Enter the terminal )
Enter the no of items 12
Enter the no of entries in item 0 5
Enter the terminal/non-terminal i
Enter the operation - S:Shift, R:Reduce S
Enter the step 5
Enter the terminal/non-terminal (
Enter the operation - S:Shift, R:Reduce S
Enter the step 4
Enter the terminal/non-terminal E
Enter the operation - S:Shift, R:Reduce -
Enter the step 1
Enter the terminal/non-terminal T
Enter the operation - S:Shift, R:Reduce -
Enter the step 2
Enter the terminal/non-terminal F
Enter the operation - S:Shift, R:Reduce -
Enter the step 3
Enter the no of entries in item 1 1
Enter the terminal/non-terminal +
Enter the operation - S:Shift, R:Reduce S
Enter the step 6
Enter the no of entries in item 2 4
Enter the terminal/non-terminal +
Enter the operation - S:Shift, R:Reduce R
Enter the step 2
Enter the terminal/non-terminal *
Enter the operation - S:Shift, R:Reduce S
Enter the step 7
Enter the terminal/non-terminal )
Enter the operation - S:Shift, R:Reduce R
```

Enter the step 2
Enter the terminal/non-terminal $
Enter the operation - S:Shift, R:Reduce R
Enter the step 2
Enter the no of entries in item 3 4
Enter the terminal/non-terminal +
Enter the operation - S:Shift, R:Reduce R
Enter the step 4
Enter the terminal/non-terminal *
Enter the operation - S:Shift, R:Reduce R
Enter the step 4
Enter the terminal/non-terminal)
Enter the operation - S:Shift, R:Reduce R
Enter the step 4
Enter the terminal/non-terminal $
Enter the operation - S:Shift, R:Reduce R
Enter the step 4
Enter the no of entries in item 4 5
Enter the terminal/non-terminal i
Enter the operation - S:Shift, R:Reduce S
Enter the step 5
Enter the terminal/non-terminal (
Enter the operation - S:Shift, R:Reduce S
Enter the step 4
Enter the terminal/non-terminal E
Enter the operation - S:Shift, R:Reduce -
Enter the step 8
Enter the terminal/non-terminal T
Enter the operation - S:Shift, R:Reduce -
Enter the step 2
Enter the terminal/non-terminal F
Enter the operation - S:Shift, R:Reduce -
Enter the step 3
Enter the no of entries in item 5 4
Enter the terminal/non-terminal +
Enter the operation - S:Shift, R:Reduce R
Enter the step 6
Enter the terminal/non-terminal *
Enter the operation - S:Shift, R:Reduce R
Enter the step 6
Enter the terminal/non-terminal)
Enter the operation - S:Shift, R:Reduce R
Enter the step 6
Enter the terminal/non-terminal $
Enter the operation - S:Shift, R:Reduce R
Enter the step 6
Enter the no of entries in item 6 4
Enter the terminal/non-terminal i

Enter the operation - S:Shift, R:Reduce S
Enter the step 5
Enter the terminal/non-terminal (
Enter the operation - S:Shift, R:Reduce S
Enter the step 4
Enter the terminal/non-terminal T
Enter the operation - S:Shift, R:Reduce -
Enter the step 9
Enter the terminal/non-terminal F
Enter the operation - S:Shift, R:Reduce -
Enter the step 3
Enter the no of entries in item 7 3
Enter the terminal/non-terminal i
Enter the operation - S:Shift, R:Reduce S
Enter the step 5
Enter the terminal/non-terminal (
Enter the operation - S:Shift, R:Reduce S
Enter the step 4
Enter the terminal/non-terminal F
Enter the operation - S:Shift, R:Reduce -
Enter the step 10
Enter the no of entries in item 8 2
Enter the terminal/non-terminal +
Enter the operation - S:Shift, R:Reduce S
Enter the step 6
Enter the terminal/non-terminal)
Enter the operation - S:Shift, R:Reduce S
Enter the step 11
Enter the no of entries in item 9 4
Enter the terminal/non-terminal +
Enter the operation - S:Shift, R:Reduce R
Enter the step 1
Enter the terminal/non-terminal *
Enter the operation - S:Shift, R:Reduce S
Enter the step 7
Enter the terminal/non-terminal)
Enter the operation - S:Shift, R:Reduce R
Enter the step 1
Enter the terminal/non-terminal $
Enter the operation - S:Shift, R:Reduce R
Enter the step 1
Enter the no of entries in item 10 4
Enter the terminal/non-terminal +
Enter the operation - S:Shift, R:Reduce R
Enter the step 3
Enter the terminal/non-terminal *
Enter the operation - S:Shift, R:Reduce R
Enter the step 3

```
Enter the terminal/non-terminal )
Enter the operation - S:Shift, R:Reduce R
Enter the step 3
Enter the terminal/non-terminal $
Enter the operation - S:Shift, R:Reduce R
Enter the step 3
Enter the no of entries in item 11 4
Enter the terminal/non-terminal +
Enter the operation - S:Shift, R:Reduce R
Enter the step 5
Enter the terminal/non-terminal *
Enter the operation - S:Shift, R:Reduce R
Enter the step 5
Enter the terminal/non-terminal )
Enter the operation - S:Shift, R:Reduce R
Enter the step 5
Enter the terminal/non-terminal $
Enter the operation - S:Shift, R:Reduce R
Enter the step 5
E       T       F       i       +       *       (       )       $
-1      -2      -3      S5                      S4
                                S6
                                R2      S7              R2      R2
                                R4      R4              R4      R4
-8      -2      -3      S5                      S4
                                R6      R6              R6      R6
        -9      -3      S5                      S4
                -10     S5                      S4
                                S6                      S11
                                R1      S7              R1      R1
                                R3      R3              R3      R3
                                R5      R5              R5      R5
Enter the no of productions 6
E->E+T
E->T
T->T*F
T->F
F->(E)
F->i
Enter the input string i+i*i
Stack                   Input
$0                      i+i*i$
$0i5                    +i*i$
$0F3                    +i*i$
$0T2                    +i*i$
$0E1                    +i*i$
$0E1+6                  i*i$
$0E1+6i5                *i$
```

```
$0E1+6F3          *i$
$0E1+6T9          *i$
$0E1+6T9*7         i$
$0E1+6T9*7i5       $
$0E1+6T9*7F10      $
$0E1+6T9           $
$0E1               $
String Accepted!Do you want to enter another string?1
Enter the input string i-i
Stack              Input
$0                  i-i$
$0i5                -i$
String Not accepted.Do you want to enter another string?1
Enter the input string i+
Stack          Input
$0              i+$
$0i5            +$
$0F3            +$
$0T2            +$
$0E1            +$
$0E1+6           $
String Not accepted.Do you want to enter another string?0
```

Viva Questions:

1. What is an ambiguous grammar? Give an example.
2. Construct SLR parse table for the grammar
   E-> E sub R | E sup E| {E/} | c
   R->E sup E | E

Ex: No:8	Implementation of front end of a compiler that generates the three address code
Date:	

Aim:

To implement the front end of a compiler that generates the three address code for a simple language.

Program :

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
int ag=0,z=1;
void main()
{
      char
a[50],id[50],b[50],op[50],mov[]="MOVF",mul[]="MULF",div[]="DIVF",add[]="ADDF",sub[]=
"SUBF",ti=0;
        int i=0,j=0,k=0,len=0,s=0,e=0,r=1,count;
        FILE *fp;
        fp=fopen("out.txt","w");
        printf("\nEnter the code:");
        scanf("%s",a);
        strcpy(b,a);
        len=strlen(a);

        for ( i=0;i<strlen(b);i++ )
        {
                if ( b[i] == '*' || b[i] == '/' )
                {
                        for ( j=i-1;b[j]!='-'&&b[j]!='+'&&b[j]!='*'&&b[j]!='/'&&b[j]!='=';j--);
                        k=j+1;
                        count=0;
                        printf("\nt%d=",ti++);
                        for ( j=j+1;count<2&&b[j]!='\0';j++ )
                        {
                                if ( b[j+1] == '+' || b[j+1] == '-' || b[j+1] == '*' || b[j+1] == '/' )
                                        count++;
                                printf("%c",b[j]);

                        }
                        b[k++]='t';
                        b[k++]=ti-1+48;
                        for ( j=j,k=k;k<strlen(b);k++,j++ )
                                b[k]=b[j];
                        i=0;
```

```
                }
        }
        for ( i=0;i<strlen(b);i++ )
        {
                if ( b[i] == '+' || b[i] == '-' )
                {
                        for ( j=i-1;b[j]!='-'&&b[j]!='+'&&b[j]!='=';j--);
                        k=j+1;
                        count=0;
                        printf("\nt%d=",ti++);
                        for ( j=j+1;count<2&&b[j]!='\0';j++ )
                        {
                                if ( b[j+1] == '+' || b[j+1] == '-' )
                                        count++;
                                printf("%c",b[j]);
                        }
                        b[k++]='t';
                        b[k++]=ti-1+48;
                        for ( j=j,k=k;k<strlen(b);k++,j++ )
                                b[k]=b[j];
                }
        }
        printf("\n%s",b);
}
```

OUTPUT :
$ cc front.c
$ ./a.out

Enter the code:c=a+b

t0=a+b
c=t0

$ ./a.out

Enter the code:c=a+b*c/d

t0=b*c
t1=t0/d
t2=a+t1
c=t2

Viva Questions:

1. What is the use of Dependency graph?
2. What is L-attributed and S-attributed definitions?
3. Write the algorithm to construct the dependency graph for the given parse tree.
4. Define S-attributed definition.
5. When you say that an SDD is L-attributed

Ex: No:9	Implementation of the back end of the compiler
Date:	

Aim:

To implement the backend of the complier which generate the assembly code.

Program:

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
int ag=0,z=1;
void main()
{
	char
a[50],id[50],mov[]="MOVF",mul[]="MULF",div[]="DIVF",add[]="ADDF",sub[]="SUBF";
	int i=0,j=0,len=0,s=0,e=0,r=1;
	FILE *fp;
	fp=fopen("out.txt","w");
	printf("\nEnter the code:");
	gets(a);
	len=strlen(a);
	for(i=0;i<len;i++)
	{
		if(a[i]=='=')
		{
			for(j=i;j<len;j++)
				if(a[j]=='i')
				{
					fprintf(fp,"\n%s ",mov);
					fprintf(fp,"%c%c%c,R%d",a[j],a[j+1],a[j+2],r++);
				}
		}
		else if((a[i]<=57)&&(a[i]>=48))
		if((a[i+1]<=57)&&(a[i+1]>=48))
			fprintf(fp,"\n%s #%c%c,R%d",mov,a[i],a[i+1],r++);
	}
	for(i=len-1;i>=0;i--)
	{
		if(a[i]=='+')
		{
			fprintf(fp,"\n%s ",add);
			e=a[i-1];
			e--;
			s=e;
```

```
                if(a[i+1]=='i')
                        fprintf(fp,"R%c,R%d",e,r-1);
                }
                else if(a[i]=='-')
                {
                        fprintf(fp,"\n%s ",sub);
                        e=a[i-1];
                        e--;
                        s=e;
                        if(a[i+1]=='i')
                                fprintf(fp,"R%c,R%c",(a[i+3]-1),s);
                        else
                                fprintf(fp,"R%c,R%d",e,r-1);
                }
                else if(a[i]=='*')
                {
                        fprintf(fp,"\n%s ",mul);
                        e=a[i-1];
                        e--;
                        s=e;
                        if(a[i+1]=='i')
                                fprintf(fp,"R%c,R%c",(a[i+3]-1),s);
                        else
                                fprintf(fp,"R%c,R%d",e,r-1);
                }
                else if(a[i]=='/')
                {
                        fprintf(fp,"\n%s ",div);
                        e=a[i-1];
                        e--;
                        s=e;
                        if(a[i+1]=='i')
                                fprintf(fp,"R%c,R%c",(a[i+3]-1),s);
                        else
                                fprintf(fp,"R%c,R%d",e,r-1);
                }
        }
    fprintf(fp,"\n%s R1,id1",mov);
}
```

OUTPUT :

$ cc back.c
$ ./a.out

Enter the code:id1=id2+id3*id4

out.txt
MOVF id2,R1
MOVF id3,R2
MOVF id4,R3
MULF R3,R2
ADDF R1,R3
MOVF R1,id1

Viva Questions:

1. Define synthesized attributes and inherited attributes.
2. List the different approaches in parameter passing.
3. Write the different memory allocation strategies.
4. How would you represent the following equation using the DAG, a: =b*c + b*c. What is the purpose of DAG?
5. What is the intermediate code representation for the expression a or b and not c ?

Ex: No:10	Implementation of Code optimization
Date:	

Aim:

To implement the code optimization of compiler using C programming.

PROGRAM :

```
#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,*tem,temp,t;
        char nu[]="\0";
        printf("\nEnter the no of values:");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("\nLeft ");
                scanf("%s",&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++)
        {
                temp=op[i].l;
                p=NULL;
                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++;
                        break;
                }
        }
}

printf("\nAfter dead code elimination\n");
for(k=0;k<z;k++)
        printf("%c\t=%s\n",pr[k].l,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)
                {
                        pr[j].l=pr[m].l;
                        for(i=0;i<z;i++)
                        {
                                if(l)
                                {
                                        a=l-pr[i].r;
                                        pr[i].r[a]=pr[m].l;
                                }
                        }
                }
        }
}

printf("\nEliminate common expression\n");
for(i=0;i<z;i++)
        printf("%c\t=%s\n",pr[i].l,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,nu);
                }
        }
```

```
}

printf("\nOptimized code\n");
for(i=0;i<z;i++)
        if(pr[i].l!='\0')
                printf("%c\t=%s\n",pr[i].l,pr[i].r);

}
```

Output :

```
$ cc codeop.c
$ ./a.out

Enter the no of values:5

Left a
Right 10

Left b
Right 20

Left c
Right a+b

Left d
Right a+b

Left e
Right c+d

Intermediate code
a=10
b=20
c=a+b
d=a+b
e=c+d

After dead code elimination
a       =10
b       =20
c       =a+b
d       =a+b
```

Eliminate common expression

```
a    =10
b    =20
c    =a+b
d    =a+b
```

Optimized code

```
a    =10
b    =20
c    =a+b
```

Viva Questions:

1. Let A be a 10x20 array with low1=low2=1. Therefore n1=10 and n2=20. Take w to be 4.
2. Give the annotated parse tree for the assignment x:=A[y,z]
3. Construct the dag for the following basic block:
   d: = b * c
   e: = a +b
   b: = b * c
   a: = e d
4. What are called optimizations and what is an optimization compiler?
5. Mention the criteria for code improving transformations.