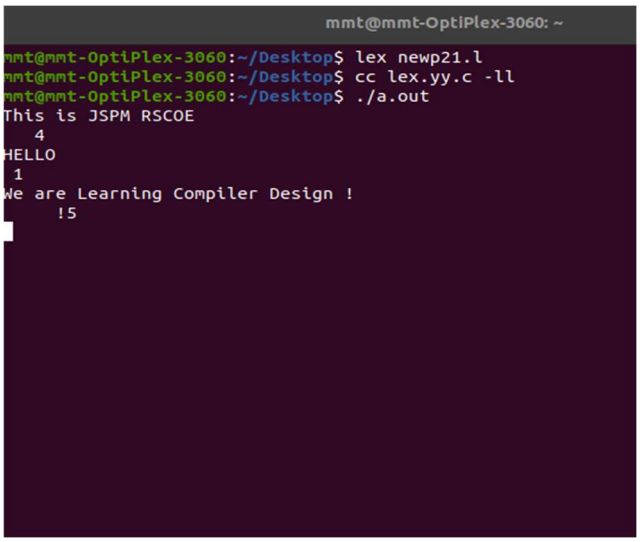
Program 1: Write a program to count number of lines, tabs, spaces, words, characters from a given text file.

Input:

```
%{
#include<stdio.h>
#include<string.h>
int i = 0;
%}

/* Rules Section*/
%%
([a-zA-Z0-9])*    {i++;} /* Rule for counting
                    number of words*/
"\n" {printf("%d\n", i); i = 0;}
%%

int yywrap(void){}
int main()
{
   // The function that starts the analysis
   yylex();
   return 0;
}
```

Output:



Program 2: Lex code to count total number of tokens.

Input :

```
%{
int n = 0 ;
%}

%%
"while"|"if"|"else" {n++;printf("\t keywords : %s", yytext);}
"int"|"float" {n++;printf("\t keywords : %s", yytext);}

[a-zA-Z_][a-zA-Z0-9_]* {n++;printf("\t identifier : %s", yytext);}

"<="|"=="|"="|"++"|"-"|"*"|"+" {n++;printf("\t operator : %s", yytext);}

[(){}|, ;] {n++;printf("\t separator : %s", yytext);}
[0-9]*"."[0-9]+ {n++;printf("\t float : %s", yytext);}

[0-9]+ {n++;printf("\t integer : %s", yytext);}

. ;
%%
int main()
{
        yylex();
        printf("\n total no. of token = %d\n", n);
}
```

Output :

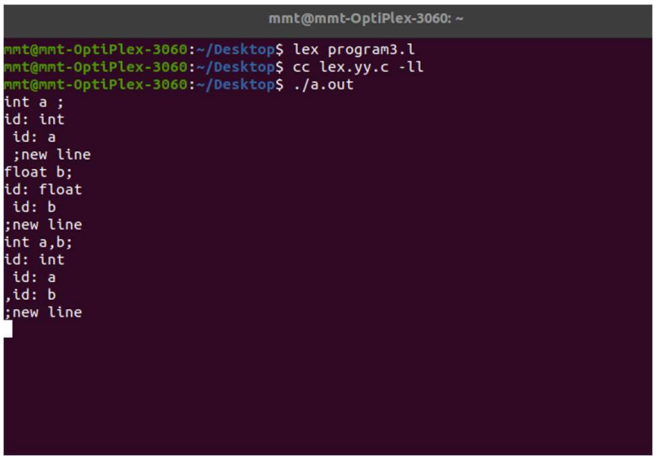
```
user@user:~$ lex 4.l
user@user:~$ cc lex.yy.c -lfl
user@user:~$ ./a.out
int p=1,d=0,r=4;
        keywords : int  identifier : p  operator : =    integer : 1     separator :
,   identifier : d  operator : =    integer : 0     separator : ,   identifier : r  o
perator : =    integer : 4     separator : ;
float m=0.0,n=200.0;
        keywords : float        identifier : m  operator : =    float : 0.0    separ
ator : ,    identifier : n  operator : =    float : 200.0   separator : ;
while(p<=3)
        keywords : while        separator : (   identifier : p  operator : <=  integ
er : 3      separator : )
{
        separator : {
if(d==0)
        keywords : if   separator : (   identifier : d  operator : ==   integer : 0
     separator : )
m=m+n*r+4.5; d++;
        identifier : m  operator : =    identifier : m  operator : +    identifier :
 n  operator : *    identifier : r  operator : +    float : 4.5     separator : ;   i
dentifier : d  operator : ++   separator : ;
else
        keywords : else
r++; m=m+r+1000.0;
        identifier : r  operator : ++   separator : ;   identifier : m  operator : =
    identifier : m  operator : +    identifier : r  operator : +    float : 1000.0  s
eparator : ;
p++;
        identifier : p  operator : ++   separator : ;
}
        separator : }

 total no. of token = 64
```

Program 3 : Write a program for syntax checking of subset of given language using LEX and YACC.

Input :

```
digit    [0-9]
letter   [a-zA-Z]
%{
# include<stdio.h>
%}
%%
{letter}({letter}|{digit})*          {printf("id: %s\n", yytext);}
\n                                          {printf("new line\n");}
%%
int main() {
        yylex();
}
int  yywrap(void) {
return 1;
}
```

Output:



Program : Write a program for syntax checking of control statements using LEX and YACC.

Input :

Exper4.l

```
%{
#include<stdio.h>
#include "y.tab.h"
%}
L [A-Za-z]
D [0-9]
id {L}({L}|{D})*
%%
"if" {return IF;}
"else" {return ELSE;}
"for"  {return FOR;}
"do"  {return DO;}
"while" {return WHILE;}
"++" {return INC;}
"--" {return DEC;}
"||"  {return OR;}
"&&" {return AND;}
"!"  {return NOT;}
"switch" {return SWITCH;}
"case" {return CASE;}
"break" {return BREAK;}
"default" {return DEFAULT;}
[0-9]+(\.[0-9]+)?  {return NUM;}
{id} {return id;}
"<"|"<="|">"|">="|"=="|"!="  {return relop;}
[-/;=+*,\(\)\{\}:]   {return yytext[0];}
[ ] {}
\n {}
%%

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

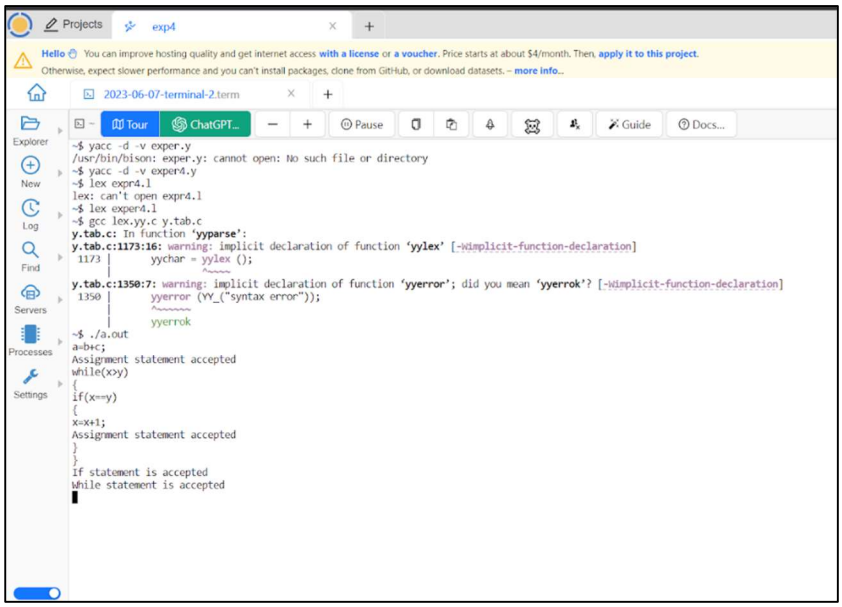
Exper4.y

```
%{
#include <stdio.h>
%}
%token id NUM OR AND NOT relop TRUE FALSE INC DEC IF ELSE DO WHILE
uminus FOR SWITCH CASE BREAK DEFAULT
%right '='
%left '+' '-'
```

```
%left '*' '/'
%right '^'
%nonassoc uminus
%left OR
%left AND
%nonassoc NOT
%%

S1 : S1 S
   | S
   ;
 S : AS ';' {printf("Assignment statement accepted \n");}
   |IFS     {printf("If statement is accepted \n");}
   |IFES    {printf("If else statement is accepted\n");}
   |WS      {printf("While statement is accepted\n");}
   |DWS     {printf("Do while statement is accepted\n");}
   |FORS    {printf("For statement is accepted\n");}
   |SS        {printf("Switch statement is accepted");}
   ;
SS : SWITCH('E')"{' CV '}'
;
CV : CASE E ':' S1 BREAK ';'
   | CASE E ':' S1 BREAK ';' CV
   |  CASE E ':' S1 BREAK ';'  DEFAULT ':' S1
   ;
AS :  id '=' E
   ;
E : E'+'E
  |E'-'E
  |E'*'E
  |E'/'E
  |E'^'E
  |'-' E %prec uminus
  |id
  |NUM
  ;
IFS: IF'('BE')"{'S1'}'
   ;
BE : BE OR BE
   | BE AND BE
   | NOT BE
   |id relop id
   |TRUE
   |FALSE
   ;
IFES : IF'('BE')"{'S1'}'ELSE'{'S1'}'
     ;
WS : WHILE '('BE')"{'S1'}'
   ;
DWS : DO '{'S1'}'WHILE'('BE')";'
```

```
   ;
FORS : FOR('IS';'BE';'MS')"{'S1'}'
     ;
IS : AS
   | IS ',' AS
   ;
MS : IS
   | id INC
   | INC id
   | id DEC
   | DEC id
   ;
%%
void main()
{
yyparse();
}
int yyerror(char *msg)
{
 printf("%s\n",msg);
}
```

Output :



Program : Write a program to check syntax of declaration statement using LEX and YACC.

Input :

Decl.l

```
%{
# include<stdio.h>
# include "y.tab.h"
%}
L  [A-Za-z]
D  [0-9]
id  {L}({L}|{D})*
%%
"int"   {return INT;}
[0-9]+  {return INUM;}
{id}    {return id;}
[,;=]   {return yytext[0];}
\n   {}
%%
int yywrap()
{
 return 1;
 }
```

Decl.y

```
%{
#include <stdio.h>
#include <stdlib.h> // Added for the exit() function
%}

%start S
%token id INT INUM
```

```
%%

S : S DL   { printf("Declaration accepted\n"); }
  | DL     { printf("Declaration accepted\n"); }
  ;

DL : INTV ';'
   ;

INTV : INT IV
     ;

IV : I
   | IV ',' id
   | IV ',' id '=' INUM
   | id '=' INUM
   ;

I : id
  ;

%%

int main()
{
    printf("Enter Valid Declaration\n");
    yyparse();
    return 0;
}

int yyerror(char *msg)
{
    printf("Invalid Statement\n");
    printf("%s\n", msg);
```

```
    exit(1); // Changed square brackets to parentheses for exit() function
}
```

Output:

```
~$ yacc -d -v decl.y
~$ lex decl.l
~$ gcc lex.yy.c y.tab.c
y.tab.c: In function 'yyparse':
y.tab.c:1031:16: warning: implicit declaration of function 'yylex' [-
 1031 |         yychar = yylex ();
      |                  ^~~~~
y.tab.c:1178:7: warning: implicit declaration of function 'yyerror';
function-declaration]
 1178 |         yyerror (YY_("syntax error"));
      |         ^~~~~~~
      |         yyerrok
~$ ./a.out
Enter Valid Declaration
int a=5;
 Declaration accepted
a=5;
Invalid Statement
syntax error
~$
```

Program : Implement a desk calculator using LEX and YACC.

Input :

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;
}
```

Command for Run Code:

1. lex calc.l
2. yacc -d calc.y
3. gcc lex.yy.c y.tab.c -w
4. ./a.out

Output:

```
avinash@aviax:~/avi/CD/CD LAB/lab Assignment 6$ lex clac.l
lex: can't open clac.l
avinash@aviax:~/avi/CD/CD LAB/lab Assignment 6$ lex calc.l
avinash@aviax:~/avi/CD/CD LAB/lab Assignment 6$ yacc -d calc.y
calc.y:41 parser name defined to default :"parse"
avinash@aviax:~/avi/CD/CD LAB/lab Assignment 6$ gcc lex.yy.c y.tab.c -w
avinash@aviax:~/avi/CD/CD LAB/lab Assignment 6$ ./a.out

Enter Any Arithmetic Expression which can have operations Addition, Subtraction,
 Multiplication, Division, Modulus and Round brackets:
(5+7*4)

Result=33

Entered arithmetic expression is Valid

avinash@aviax:~/avi/CD/CD LAB/lab Assignment 6$
```

Program : Write a program to generate ICG using LEX and YACC.

Input :

Icg.l

```
%{
#include"y.tab.h"
extern char yyval;
%}

%%

[0-9]+ {yylval.symbol=(char)(yytext[0]);return NUMBER;}
[a-z] {yylval.symbol= (char)(yytext[0]);return LETTER;}
. {return yytext[0];}
\n {return 0;}

%%
```

Icg.y

```
%{
#include"y.tab.h"
#include<stdio.h>
char addtotable(char,char,char);


int index1=0;
char temp = 'A'-1;


struct expr{

char operand1;
char operand2;
char operator;
char result;
};


%}

%union{
char symbol;
}

%left '+' '-'
%left '/' '*'
```

```
%token <symbol> LETTER NUMBER
%type <symbol> exp
%%

statement: LETTER '=' exp ';' {addtotable((char)$1,(char)$3,'=');};
exp: exp '+' exp {$$ = addtotable((char)$1,(char)$3,'+');}
   |exp '-' exp {$$ = addtotable((char)$1,(char)$3,'-');}
   |exp '/' exp {$$ = addtotable((char)$1,(char)$3,'/');}
   |exp '*' exp {$$ = addtotable((char)$1,(char)$3,'*');}
   |'(' exp ')' {$$= (char)$2;}
   |NUMBER {$$ = (char)$1;}
   |LETTER {(char)$1;};

%%

struct expr arr[20];

void yyerror(char *s){
   printf("Errror %s",s);
}

char addtotable(char a, char b, char o){
   temp++;
   arr[index1].operand1 =a;
   arr[index1].operand2 = b;
   arr[index1].operator = o;
   arr[index1].result=temp;
   index1++;
   return temp;
}

void threeAdd(){

   int i=0;
   char temp='A';
   while(i<index1){
      printf("%c:=\t",arr[i].result);
      printf("%c\t",arr[i].operand1);
      printf("%c\t",arr[i].operator);
      printf("%c\t",arr[i].operand2);
      i++;
      temp++;
      printf("\n");
   }
}

void fouradd(){
   int i=0;
   char temp='A';
   while(i<index1){
```

```
        printf("%c\t",arr[i].operator);
        printf("%c\t",arr[i].operand1);
        printf("%c\t",arr[i].operand2);
        printf("%c",arr[i].result);
        i++;
        temp++;
        printf("\n");
    }

}

int find(char l){
    int i;
    for(i=0;i<index1;i++)
        if(arr[i].result==l) break;
    return i;
}


void triple(){
    int i=0;
    char temp='A';
    while(i<index1){
        printf("%c\t",arr[i].operator);
        if(!isupper(arr[i].operand1))
        printf("%c\t",arr[i].operand1);
        else{
            printf("pointer");
            printf("%d\t",find(arr[i].operand1));
        }
        if(!isupper(arr[i].operand2))
        printf("%c\t",arr[i].operand2);
        else{
            printf("pointer");
            printf("%d\t",find(arr[i].operand2));
        }
        i++;
        temp++;
        printf("\n");
    }

}


int yywrap(){
    return 1;
}

int main(){
    printf("Enter the expression: ");
```

```
    yyparse();
    threeAdd();
    printf("\n");
    fouradd();
    printf("\n");
    triple();
    return 0;
}
```

Command for run a program:

1. lex icg.l
2. yacc -d lex.y
3. gcc lex.yy.c y.tab.c -w
4. ./a.out

Output:

```
avinash@aviax:~/avi/CD/CD LAB/lab 7 ICG$ lex icg.l
avinash@aviax:~/avi/CD/CD LAB/lab 7 ICG$ yacc -d icg.y
icg.y:25 parser name defined to default :"parse"
avinash@aviax:~/avi/CD/CD LAB/lab 7 ICG$ gcc lex.yy.c y.tab.c -w
avinash@aviax:~/avi/CD/CD LAB/lab 7 ICG$ ./a.out
Enter the expression: a=b*c/d-e;
A:=     b       *       c
B:=     A       /       d
C:=     B       -       e
D:=     a       =       C

*       b       c       A
/       A       d       B
-       B       e       C
=       a       C       D

*       b       c
/       pointer0        d
-       pointer1        e
=       a       pointer2
avinash@aviax:~/avi/CD/CD LAB/lab 7 ICG$
```

Program : Write a program for code optimization.

Input :

codeOptimization.c

```
#include<stdio.h>
#include<math.h>
#include<string.h>
# include<ctype.h>
#include<stdlib.h>

// How to run
// Compile: gcc -o codeOptimization codeOptimization.c -lm
// Run: ./codeOptimization

struct quad
 {
        char ope[5];
        char arg1[5];
        char arg2[5];
        char res[5];
 }QUAD[5];
int i=0,n,c=0;

void get()
{
        printf("\nEnter no of lines in a block");
        scanf("%d",&n);
        printf("enter ICG in form operator arg1 arg2 result:");
        for(i=0;i<n;i++)
        scanf("%s\n%s\n%s\n%s",&QUAD[i].ope,&QUAD[i].arg1,&QUAD[i].arg2,&QUA
D[i].res);
}

void const_folding()
{
        int j,c1=0,d=0;
        char ch[5],ch1[5],num[10];
        int flag1 =1, flag2 =1;
                for(i=0;i<n;i++)
        {
         flag1 =1;flag2 =1;
         for (j=0;j<strlen(QUAD[i].arg1);j++)
         {
          if(!isdigit(QUAD[i].arg1[j]))

          { flag1 = 0;printf("Operand1 is not contstant, Constant folding can not applied to
quadruple %d\n",i);
           break;
```

```
            }

          }
           for (j=0;j<strlen(QUAD[i].arg2);j++)
           {
            if(!isdigit(QUAD[i].arg2[j]))

            { flag2 = 0; printf("Operand2 is not contstant, Constant folding can not applied to
quadruple %d\n",i);
              break;
            }


          }
         if(flag1 == 1 && flag2 ==1)
         {
         c=atoi(QUAD[i].arg1);
         c1=atoi(QUAD[i].arg2);

         if(strcmp(QUAD[i].ope,"*")==0)
         {
                  d=c*c1;
                  //itoa(d,ch,10);
                  snprintf(ch, 10, "%d", d);
                  strcpy(QUAD[i].ope,"=");
                  strcpy(QUAD[i].arg1,ch);
                  strcpy(QUAD[i].arg2,"\0");
         }

         if(strcmp(QUAD[i].ope,"/")==0)
         {
                  d=c/c1;
                  //itoa(d,ch,10);
                  snprintf(ch, 10, "%d", d);
                  strcpy(QUAD[i].ope,"=");
                  strcpy(QUAD[i].arg1,ch);
                  strcpy(QUAD[i].arg2,"\0");
         }

         if(strcmp(QUAD[i].ope,"+")==0)
         {
                  d=c+c1;
                  //itoa(d,ch,10);
                  snprintf(ch, 10, "%d", d);
                  strcpy(QUAD[i].ope,"=");
                  strcpy(QUAD[i].arg1,ch);
                  strcpy(QUAD[i].arg2,"\0");
         }

         if(strcmp(QUAD[i].ope,"-")==0)
```

```
        {
                d=c-c1;
                //itoa(d,ch,10);
                snprintf(ch, 10, "%d", d);
                strcpy(QUAD[i].ope,"=");
                strcpy(QUAD[i].arg1,ch);
                strcpy(QUAD[i].arg2,"\0");
        }
        }
        }


}

void strength_reduction()
{
        int j=0,n1=0,m=0,c=0,tempo=0,t=0;
        char ch[5],cc[5],ct[2],pres[5];
        int flag;
        strcpy(ct,"s");
        for(i=0;i<n;i++){
        c=0;
        if(strcmp(QUAD[i].ope,"*")==0||strcmp(QUAD[i].ope,"/")==0)
        {   j = 1;
           if(strcmp(QUAD[i].ope,"*")==0)
           flag =0;
           else
           flag =1;
        if((atoi(QUAD[i].arg2))>0)
                {
                m=atoi(QUAD[i].arg2);
                while(n1<=m)
                        {
                        n1=pow(2,j);
                        j++;
                        }
                j=j-2;
                n1=pow(2,j);
                c=m-n1;
                printf("number! is 2^%d + %d",j,c);
                if(c==0)
                        {
                        //itoa(j,ch,10);
                        snprintf(ch, 10, "%d", j);
                        if(flag==0)
                        strcpy(QUAD[i].ope,"<<");
                        else
                         strcpy(QUAD[i].ope,">>");
                    //  strcpy(QUAD[i].arg1,ch);
                        strcpy(QUAD[i].arg2,ch);
```

```
                    //  strcpy(QUAD[i].res,"t2");

                        }
                else

                        {
                        strcpy(pres,QUAD[i].res);
                        //itoa(j,ch,10);
                        snprintf(ch, 10, "%d", j);
                        if(flag==0)
                        strcpy(QUAD[i].ope,"<<");
                        else
                        strcpy(QUAD[i].ope,">>");
                        strcpy(QUAD[i].arg2,ch);
//                      strcpy(QUAD[i].res,"t2");
                        i++;

                        for(t=0;t<c;t++)
                        {
                        for(j=n;j>=i;j--)
                        QUAD[j+1] = QUAD[j];
                        if(c==1)
                        {
                        //itoa(c,ch,10);
                        snprintf(ch, 10, "%d", j);
                        if(flag==0)
                        strcpy(QUAD[i].ope,"+");
                        else
                        strcpy(QUAD[i].ope,"-");
                        tempo=i-1;
                        strcpy(QUAD[i].arg1,QUAD[tempo].res);
                        strcpy(QUAD[i].arg2,ch);
                        //itoa(i,cc,10);
                        snprintf(cc, 10, "%d", i);
                        strcat(ct,cc);
                        printf("CT is %s",ct);
                        strcpy(QUAD[i].res,ct);
                        }
                        else
                        {
                        strcpy(ct,"s");
                        //itoa(c-(c-1),ch,10);
                        snprintf(ch, 10, "%d", c-(c-1));

                        if(flag==0)
                        strcpy(QUAD[i].ope,"+");
                        else
                        strcpy(QUAD[i].ope,"-");
                        tempo=i-1;
                        strcpy(QUAD[i].arg1,QUAD[tempo].res);
```

```
                        strcpy(QUAD[i].arg2,ch);
                        //strcat("t",i);
                        //itoa(i,cc,10);
                        snprintf(cc, 10, "%d", i);
                        strcat(ct,cc);
                        strcpy(QUAD[i].res,ct);
                        }
                        i++;
                        n=n+1;
                        }
                        }

                }

        }
        printf("n value =%d\n",n);
        for(j=i;j<n;j++)
        {
        if(strcmp(QUAD[j].arg1, pres) ==0)
        strcpy(QUAD[j].arg1,QUAD[i-1].res);
        else if (strcmp(QUAD[j].arg2, pres) ==0)
        strcpy(QUAD[j].arg2,QUAD[i-1].res);
        }
        if(c!=0)
        i = i-1;

}
}

void disp()
{
printf("\nQuadraple\noperator\targ1\targ2\tresult\n");
printf("n value is %d\n",n);
for(i=0;i<n;i++)
  printf("\t%s\t%s\t%s\t%s\n",QUAD[i].ope,QUAD[i].arg1,QUAD[i].arg2,QUAD[i].res);
}

void main()
{
get();
disp();
const_folding();
printf("Quadruples after constant folding\n");
disp();
strength_reduction();
printf("Quadruples after strength reduction\n");
disp();
}
```

Output :

```
Activities   Terminal                                   Jun 6 10:59
mml@mml-OptiPlex-3060: ~
      |                    char *                          char (*)[5]
codeOptimization.c:26:22: warning: format '%s' expects argument of type 'char *', but argument 5 has type 'char (*)[5]' [-Wformat=]
   26 |  scanf("%s\n%s\n%s\n%s",&QUAD[i].ope,&QUAD[i].arg1,&QUAD[i].arg2,&QUAD[i].res);
      |                         ~^                                        ~~~~~~~~~~~
      |                          |                                        |
      |                          char *                                   char (*)[5]
mml@mml-OptiPlex-3060:~$ ./codeOptimization

Enter no of lines in a block 4
enter ICG in form operator arg1 arg2 result:
+ 2 1 t1
- 10 5 t2
* 10 4 t3
/ 50 5 t4

Quadraple
operator        arg1    arg2    result
n value is 4
        +       2       1       t1
        -       10      5       t2
        *       10      4       t3
        /       50      5       t4
Quadruples after constant folding

Quadraple
operator        arg1    arg2    result
n value is 4
        =       3               t1
        =       5               t2
        =       40              t3
        =       10              t4
n value =4
n value =4
n value =4
n value =4
Quadruples after strength reduction

Quadraple
operator        arg1    arg2    result
n value is 4
        =       3               t1
        =       5       t1      t2
        =       40      t1      t3
        =       10      t1      t4
mml@mml-OptiPlex-3060:~$ ^[[2~
```