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

AIM:

To implement lexical analyzer using C programming.

ALGORITHM:

STEP 1: Start
STEP 2: Declare all variables and file pointers
STEP 3: Display the input program.
STEP 4: Separate the keyword in the program and display it.
STEP 5: Display the reader files of the input program.
STEP 6: Separate the operators of the input program and display it.
STEP 7: Print the punctuation marks.
STEP 8: Print the constant that are present in the input program
STEP 9: Print the identifiers of the input program.

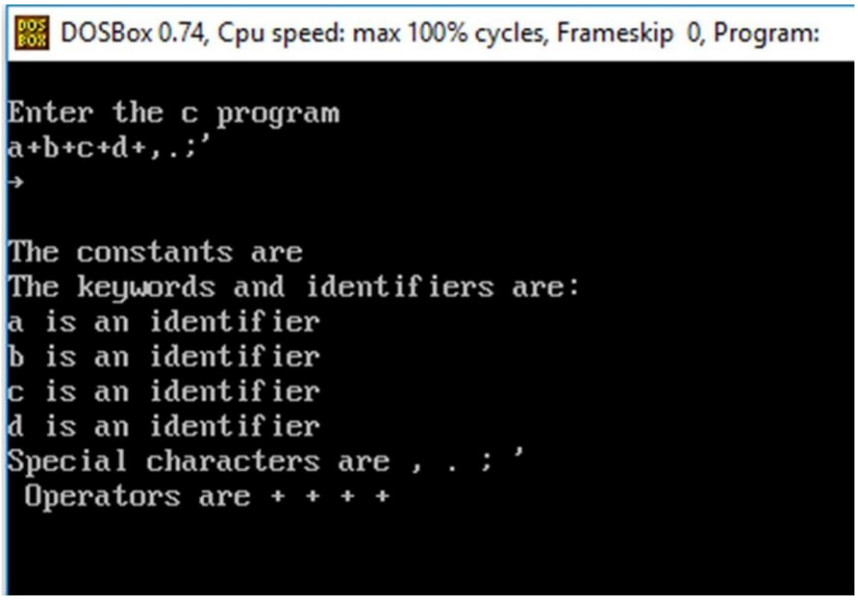
PROGRAM:

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

OUTPUT:



RESULT:

Thus, the lexical analyzer using C programming is implemented successfully.

Ex: No:4	Implementation of Symbol table
Date:	

Aim

To implement the symbol table using C programming.

Algorithm :

Step 1 : Start the program.
Step 2 : Read the input file “input.txt” in read mode.
Step 3 : Scan the entire input till eof.
1. If the string found was either int, float, double… copy into the datatype in symbol table.
2. Update the corresponding variable and value if any in symbol table.
3. If no value is in initializer the update the table as “garbage”.

Step 4 : Close the file.
Step 5 : Stop the program.

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
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 :

```
skcet@skcet-Lenovo-V110-15ISK:~/Desktop$ cc sym.c
skcet@skcet-Lenovo-V110-15ISK:~/Desktop$ ./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
skcet@skcet-Lenovo-V110-15ISK:~/Desktop$
```

Result :

Thus, the implementation of symbol table has been successfully completed using C programming.

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.

ALGORITHM:

STEP 1: Start the program

STEP 2: Obtain the high level language as input

STEP 3: Based on pattern and lexemes stored in the symbol table in the three address code is obtained

STEP 4: Three address code generated will be optimized and displayed

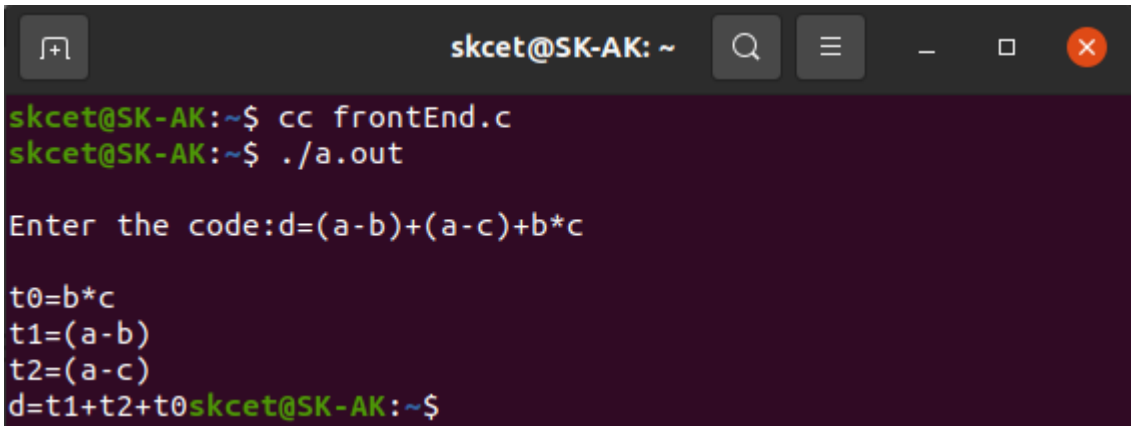
STEP 5:stop the program.

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 :



RESULT:

Thus, the implementation of the front end of a compiler that generates the three address code for a simple language is done successfully.

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.

ALGORITHM:

STEP 1: Start the program

STEP 2: Read the input with the intermediate representation

STEP 3: Based on the three address code the given input will be processed will converted to assembly code with an operation like ADD,SUB,MUL,MOV,STORE,LOAD.

STEP 4: Generated output will be returned in the file called out.txt

STEP 5: Stop the program

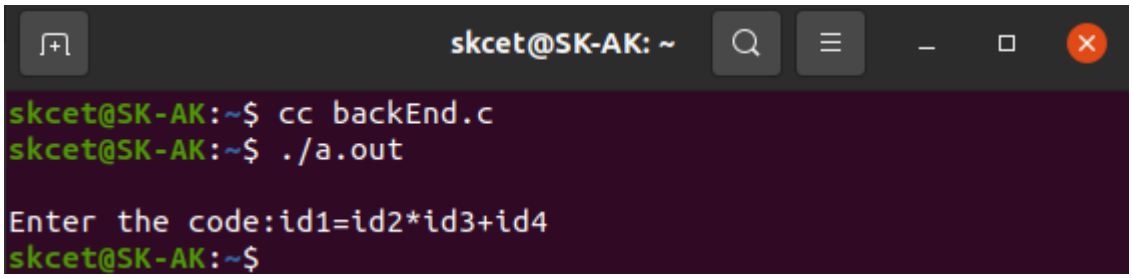
Program:

```
#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:



RESULT:

Thus, the implementation the backend of the complier which generate the assembly code is done successfully and the output is verified.

Ex: No:10	Implementation of Code optimization
Date:	

Aim:

To implement the code optimization of compiler using C programming.

ALGORITHM:

STEP 1: Start the program

STEP 2: Read the input given as assembly code

STEP 3: Apply the function preserving algorithm such as common subexpression elimination,code propagation,dead code elimination and constant folding.

STEP 4:obtain the final optimizing code for display

STEP 5:stop the program.

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 :

```
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
c          =a+b
```

```
Eliminate common expression
a          =10
b          =20
c          =a+b
c          =a+b

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

RESULT:

Thus, the implementation of the code optimization of compiler using C programming is done successfully and the output is verified.