1A **Implementation of lexical analyzer using C programming**

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.

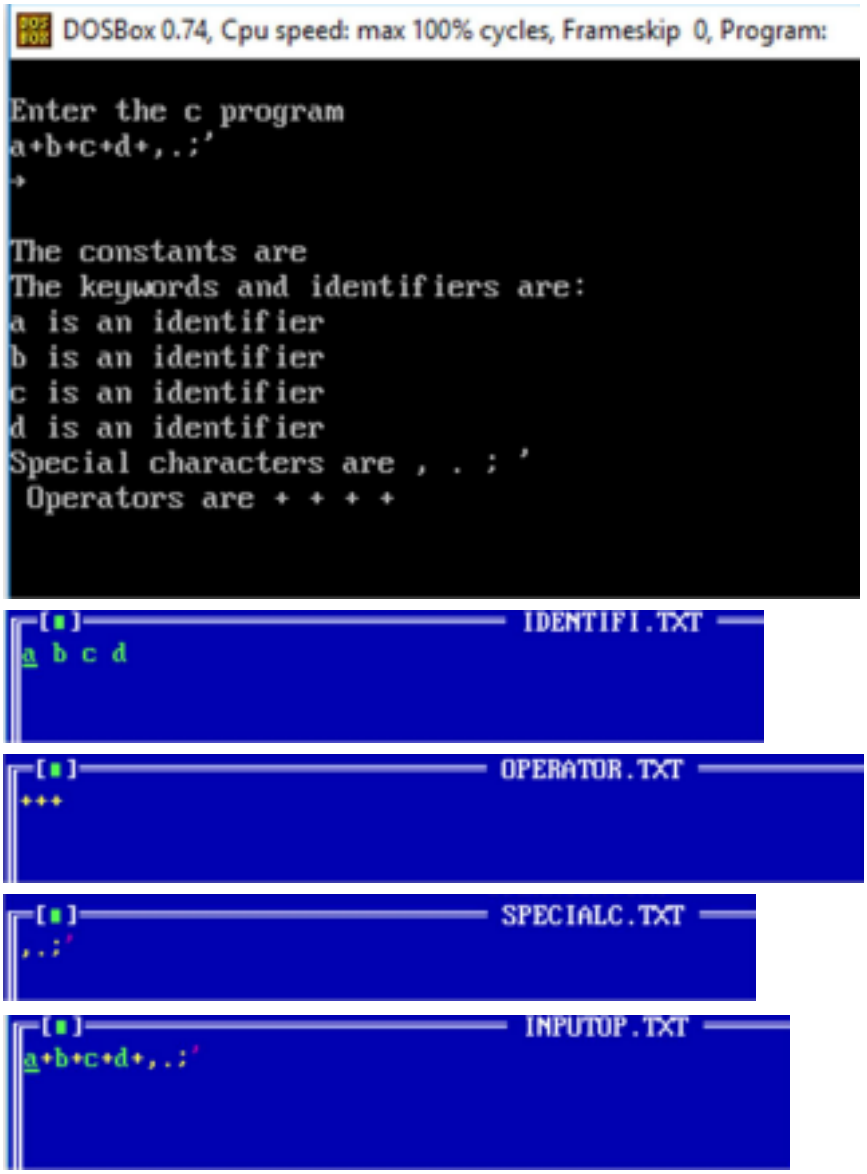
PROGRAM:

```
#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:



1B Implementation of Lexical analyzer using LEX

Step 1: Start.
Step 2: Declare all variables or header files in definition part.
Step 3: Define action and pattern in transition rule.
Step 4: Call yylex() function in main function.
Step 5: Open the text file or c.file.

Step 6: Print the number of lines, words, characters in text file.
Step 7: Print the keywords, Special characters in c file.
Step 8: Stop the program

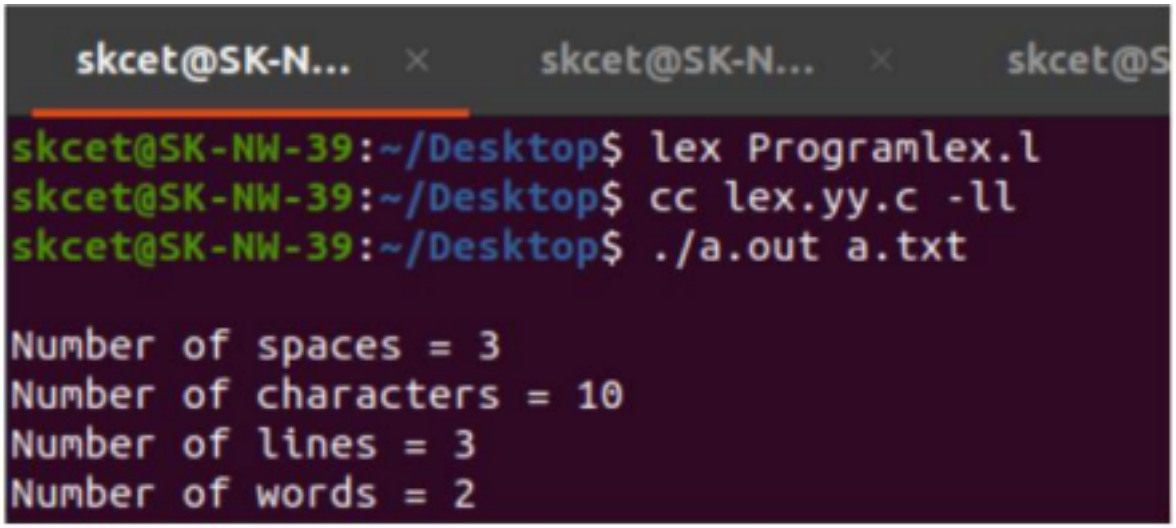
Program for count the word, character, space and line from the file using LEX:

```
%{
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 :



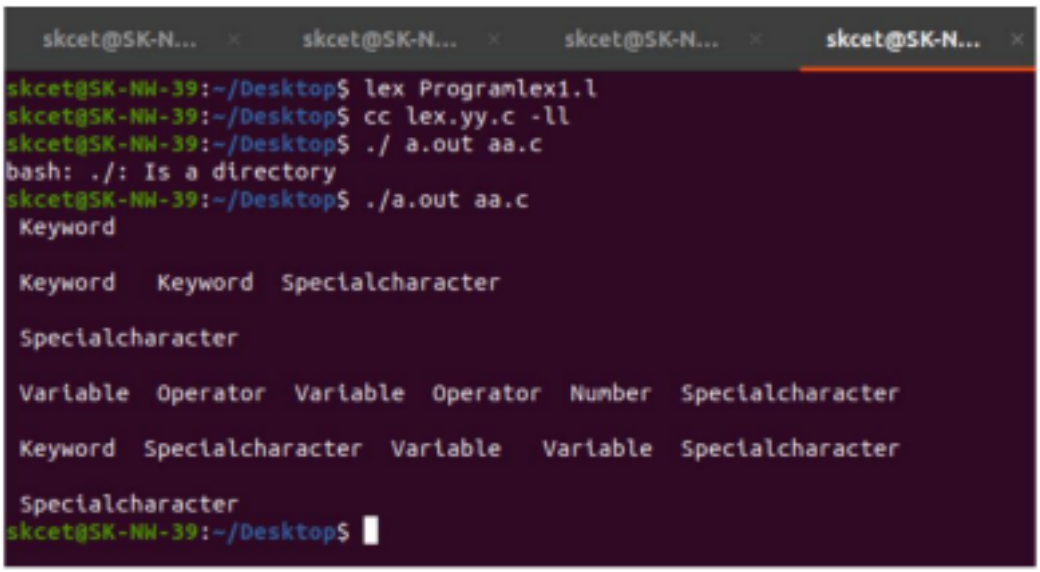
Program for Lexical Analyzer using LEX tool :

```
%{
#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|sca
nf|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 :



2 **Implementation of a calculator that takes an expression (with digits, + and *), computes and prints its value, using YACC.**

STEP 1: A YACC source program has three parts as declaration %%translation rules%%.

STEP 2: Declaration section include standard I/O header file,define global variables,define the operations and their precedence.

STEP 3: Rules section, define the rules that parse the input stream.

STEP 4: Program section contains the subroutines.

STEP 5: calc. per file include statements for standard input and output.calc.lex contain the rules to generate these tokens from input stream.

Program 1 :

//calc.l

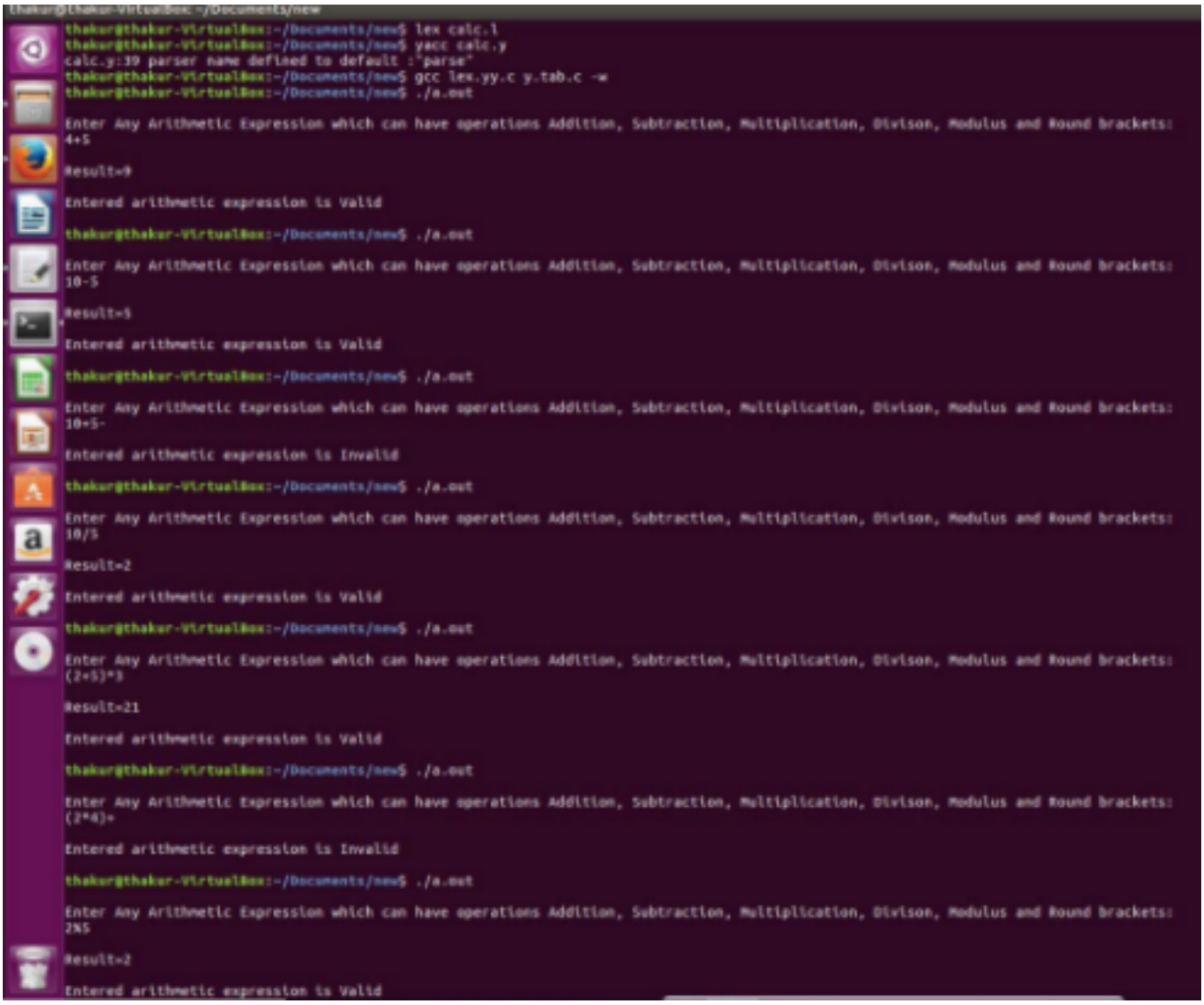
```
%{
/* Definition section */
#include<stdio.h>
#include "y.tab.h"
extern int yylval;
%}
/* Rule Section */
%%
[0-9]+ {
yylval=atoi(yytext);
return NUMBER;
}
[\t] ;
[\n] return 0;
. return yytext[0];
%%
int yywrap()
 {
return 1;
}
```

//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:



3 Implementation of Parser using LEX and YACC tool

Step 1: Start.
Step 2: Declare all variables or header files.
Step 3: Define action and pattern.
Step 4: Call the yylex() function in the main function.
Step 5: Stop.

Program :

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:

```
skcet@user-OptiPlex-380:~/exp3$ lex exp3.l
exp3.l:14: warning, rule cannot be matched
skcet@user-OptiPlex-380:~/exp3$ yacc -d e3.y
skcet@user-OptiPlex-380:~/exp3$ cc lex.yy.c y.tab.c -ll
y.tab.c: In function 'yyparse':
y.tab.c:1214:16: warning: implicit declaration of function 'yylex' [-Wimplicit-
function-declaration]
 1214 |       yychar = yylex ();
      |                ^~~~~
y.tab.c:1377:7: warning: implicit declaration of function 'yyerror'; did you me
an 'yyerrok'? [-Wimplicit-function-declaration]
 1377 |       yyerror (YY_("syntax error"));
      |       ^~~~~~~
      |       yyerrok
skcet@user-OptiPlex-380:~/exp3$ ./a.out
(id)
(Scanned a name
SUCCESS
skcet@user-OptiPlex-380:~/exp3$
```

4 Implementation of Symbol table

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;
```

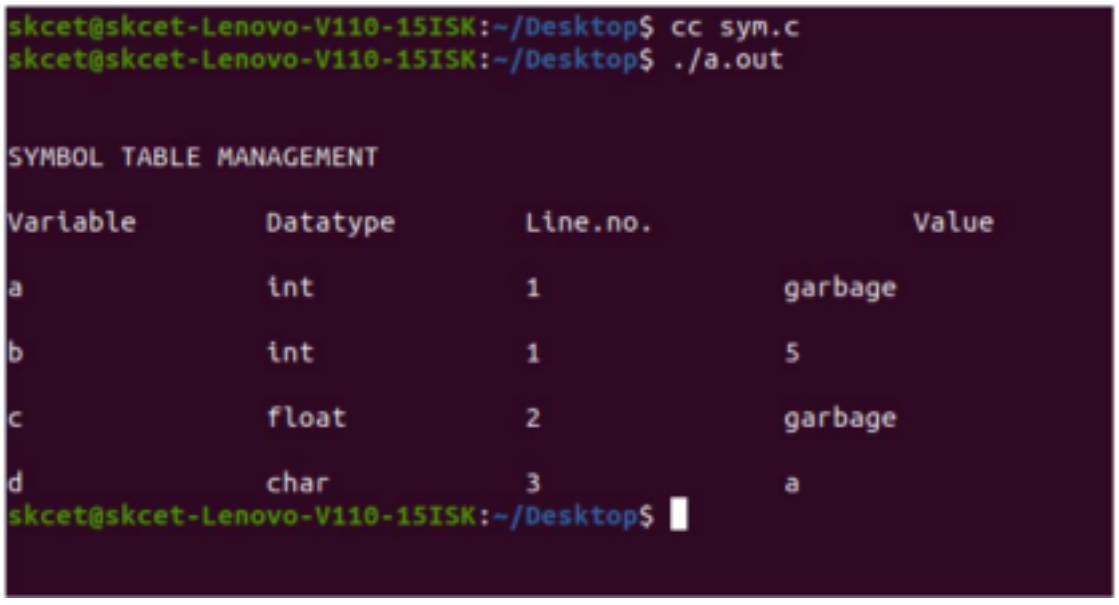
```
 char s[25],typ[25],temp[25],gar[]="garbage",t[25],got[10],e[10];
 float m2;
 FILE *fn,*ft,*fp
fn=fopen("input.txt","r");
printf("\n\nSYMBOL TABLE MANAGEMENT\n\n");
printf("Variable\tDatatype\tLine.no.\t\tValue\n");
          while(!(feof(fn)))
          {
          fscanf(fn,"%s",s);
          if((strcmp(s,"int")==0)||(strcmp(s,"float")==0))
 {
 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 :



5 **Implementation of Predictive Parser**

STEP 1: Start the program.

STEP 2: Read the input for the production rules.

STEP 3: Compute first and follow of production rules.

STEP 4: construct the predictive parsing table

STEP 5: Read the input for the string to be parsed.

STEP 6: Parse the input string with the help of table

STEP 7: Return if the string is accepted or not accepted

STEP 8: Stop the Program.

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++)
```

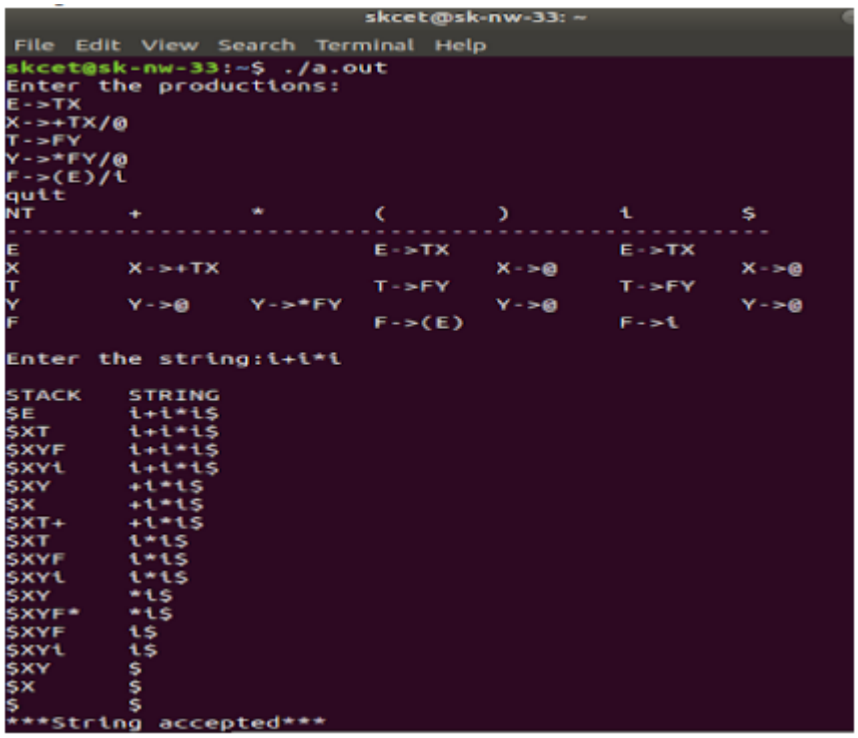
```
                {
                        if(b[x].t[i][0]==b[j].node)
                        {
                                if(b[j].k==0)
                                        first(j);
                                h=1;
                                for(k=0;k<b[j].fi;k++)
                                        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)
```

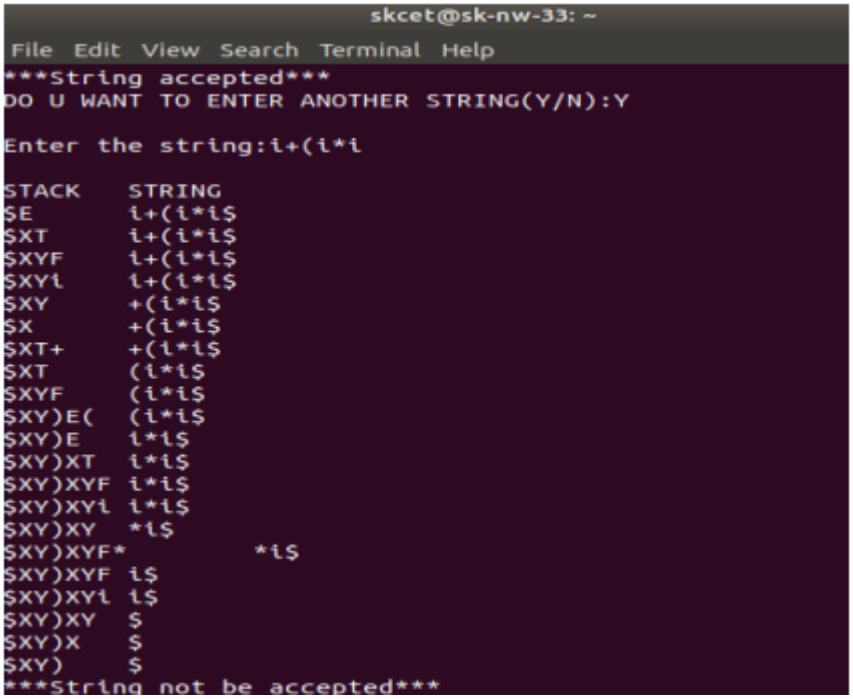
```
                                                                follow(i);

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;
```

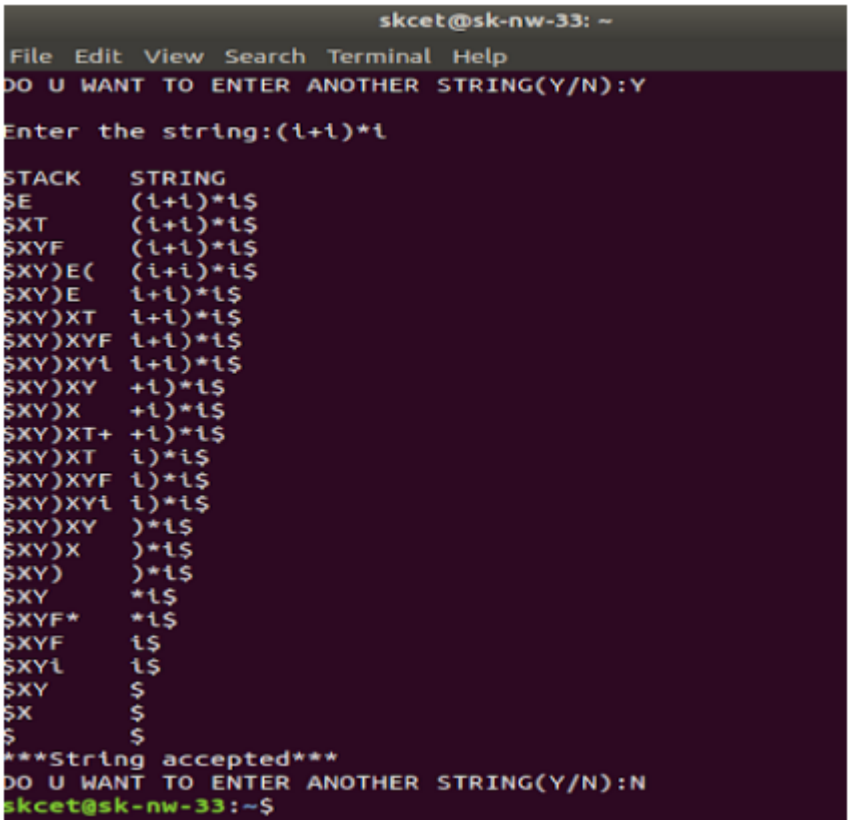
```
printf("\nEnter the string:");
scanf("%s",string);
string[strlen(string)]='$';
stack[top++]='$';
stack[top++]=b[0].node;
printf("\nSTACK\tSTRING");
while(1)
{
        printf("\n");
        for(k=0;k<top;k++)
        printf("%c",stack[k]);
        printf("\t");
        for(k=i;k<strlen(string);k++)
        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 :







6 Implementation of Shift Reduce Parser

STEP 1: Start with the sentence to be parsed as initial sentential form.

STEP 2: Shift reduce parsing uses a stack to hold the grammar and input tape holds the string.

STEP 3: Shift reduce parsing performs the two actions: shift and reduce.

STEP 4: In shift action: the current symbol in the input string is pushed to a stack.

STEP 5: In reduced action: the symbols will be replaced by non terminal. The symbol is the right side of production and non terminal is the left side of production.

STEP 6: Stop the program.

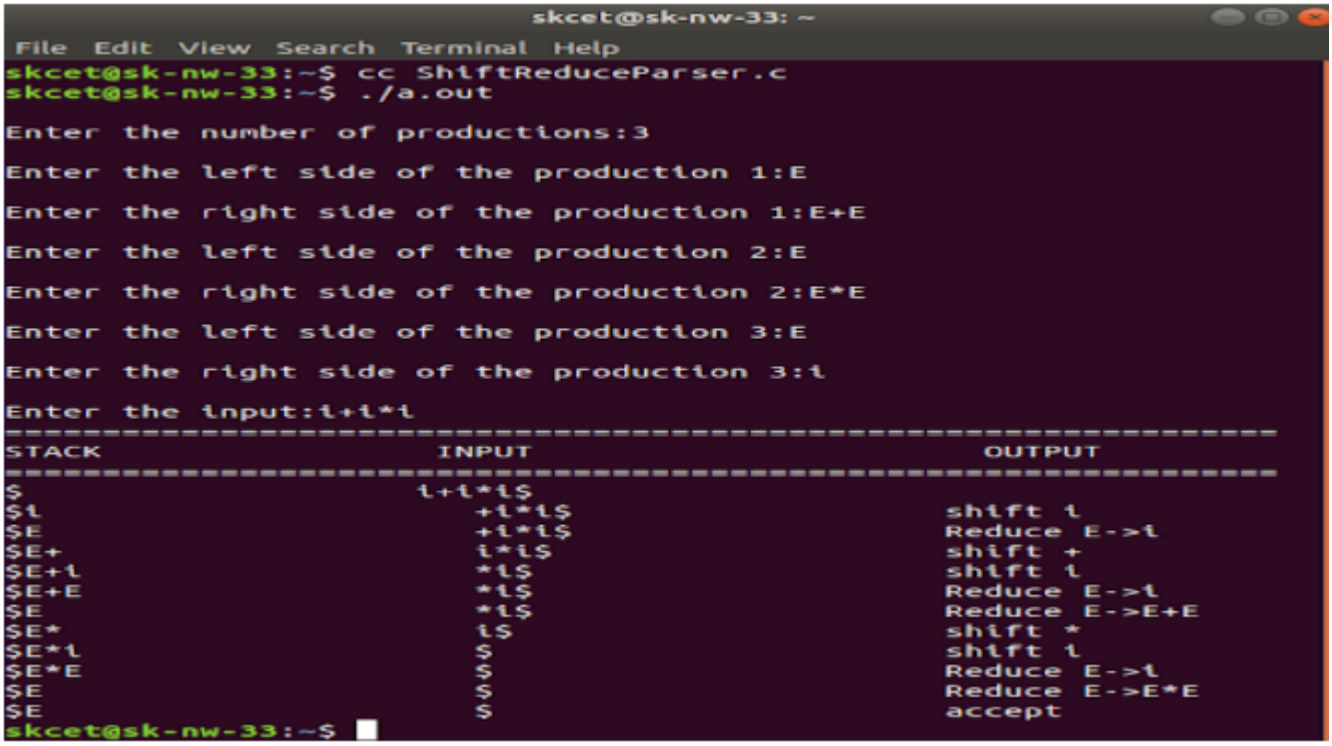
Program :

```
#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 :



7 Implementation of LR Parser

STEP 1: Write the augmented grammar by introducing a new non terminal of start symbol.
STEP 2: Form the item sets from augmented grammar.
STEP 3: Construct the SLR parsing table from the obtained item sets by using the action: Shift, Reduce, Accept And Error.

STEP 4: Obtain the stack implementation for input string: id+id*id

STEP 5: To construct SLR(1) parsing table,canonical collection of LR(0) items.

Program:

```
#include<stdio.h>
#include<string.h>
struct table
{
char op;
```

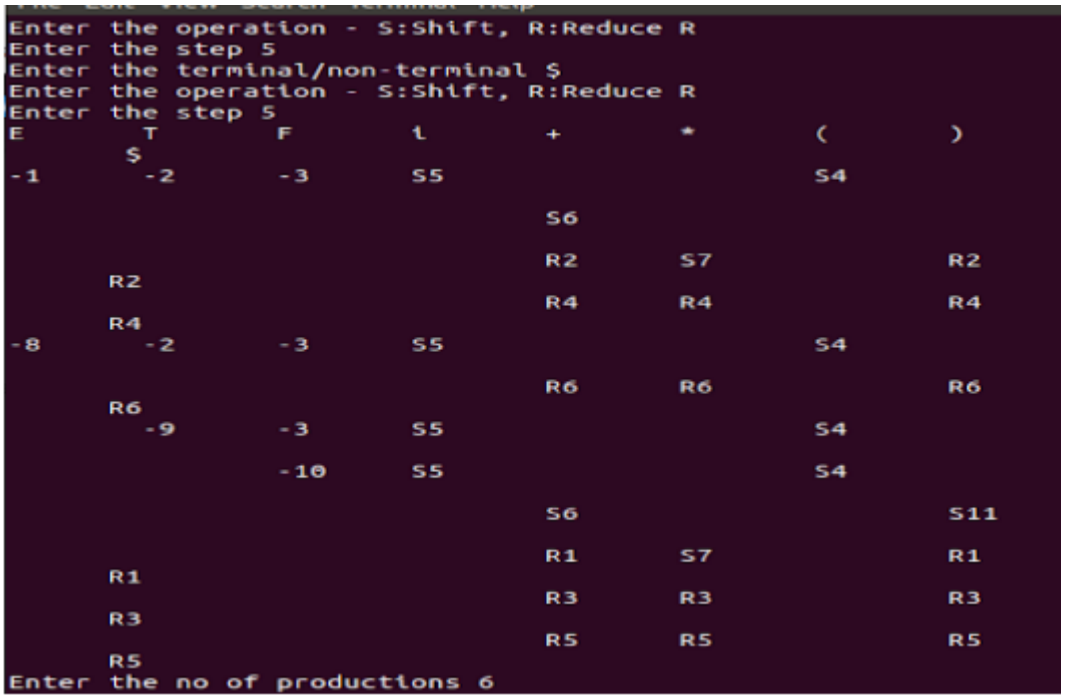
```
char set;
int value;
}ta[50][50];
void main()
{
char nt[20],t[20],temp,ip[50],s[50],prod[20][20],rhs[20][20],lhs[20][20],temparr[20]; int
non,not,i,j,l,ni,ch=1,n,tos,n1,temp1,np,no;
printf("Enter the no of non-terminals ");
scanf("%d",&non);
for ( i=0;i<non;i++ )
{
printf("Enter the non terminal ");
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++ )
```

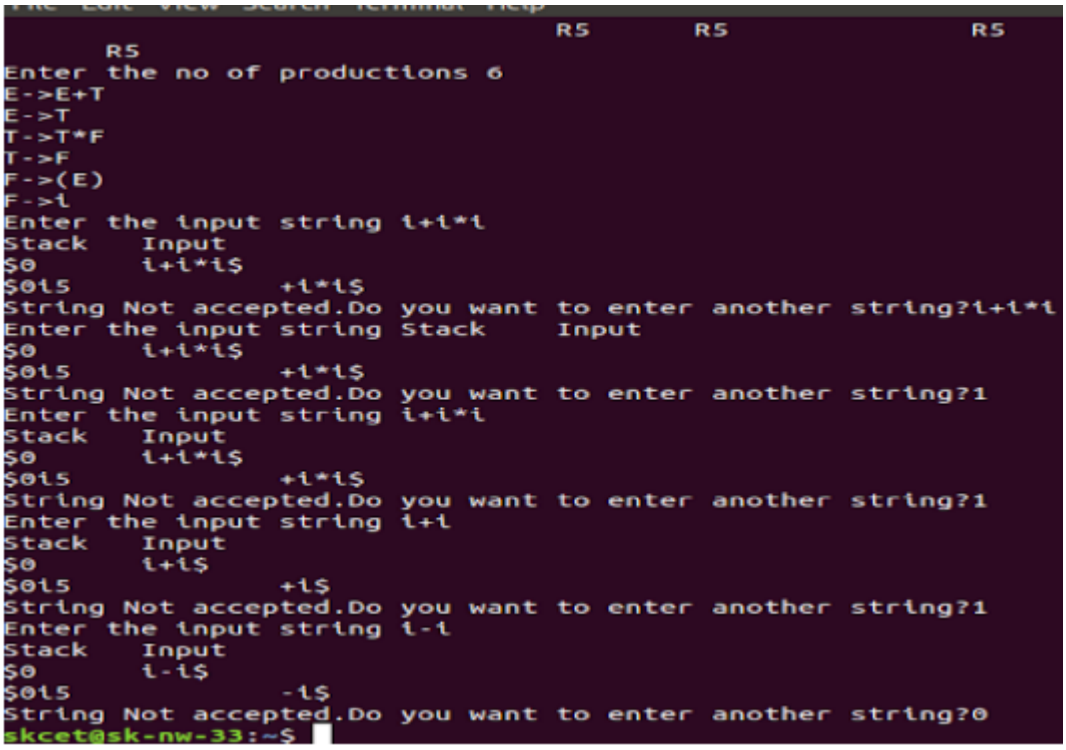
```
{
if ( ta[i][j].set == 't' )
printf("%c%d\t",ta[i][j].op,ta[i][j].value);
else
printf("\t");
}
printf("\n");
}
printf("Enter the no of productions ");
scanf("%d",&np);
for ( i=0;i<np;i++ )
{
scanf("%s",prod[i]);
lhs[i][0]=prod[i][0];
lhs[i][1]='\0';
for ( j=3;j<strlen(prod[i]);j++ )
rhs[i][j-3]=prod[i][j];
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:

```
skcet@sk-nw-33:~$ cc LRParser.c
skcet@sk-nw-33:~$ ./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
```





8 **Implementation of front end of a compiler that generates the three address code**

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 :

```
skcet@SK-AK:~$ cc frontEnd.c
skcet@SK-AK:~$ ./a.out

Enter the code:d=(a-b)+(a-c)+b*c

t0=b*c
t1=(a-b)
t2=(a-c)
d=t1+t2+t0skcet@SK-AK:~$
```

9 Implementation of the back end of the compiler

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:

```
skcet@SK-AK:~$ cc backEnd.c
skcet@SK-AK:~$ ./a.out

Enter the code:id1=id2*id3+id4
skcet@SK-AK:~$
```

out.txt

```
MOVF id2,R1
MOVF id3,R2
MOVF id4,R3
ADDF R2,R3
MULF R2,R1
MOVF R1,id1
```

10 **Implementation of Code optimization**

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 :