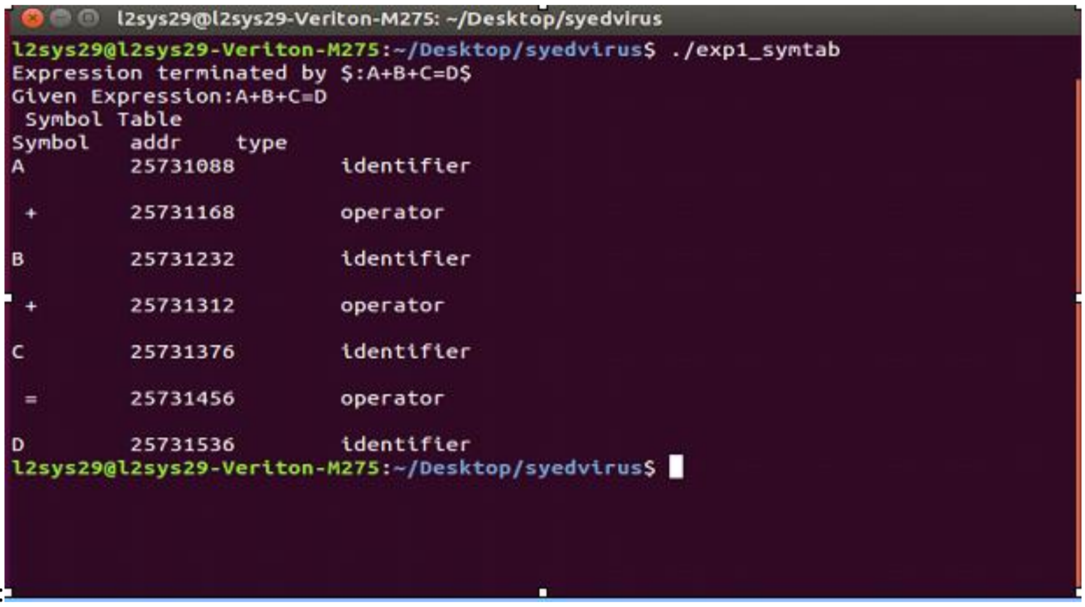
Ex.No:03

IMPLEMENTATION OF SYMBOL TABLE

Program:

```
//Implementation of symbol table
#include<stdio.h>
#include<ctype.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
void main()
{
 int i=0,j=0,x=0,n;
 void *p,*add[5];
 char ch,srch,b[15],d[15],c;
 printf("Expression terminated by $:");
 while((c=getchar())!='$')
 {
  b[i]=c;
  i++;
 }
 n=i-1;
 printf("Given Expression:");
 i=0;
 while(i<=n)
 {
  printf("%c",b[i]);
  i++;
 }
 printf("\n Symbol Table\n");
 printf("Symbol \t addr \t type");
 while(j<=n)
 {
  c=b[j];
  if(isalpha(toascii(c)))
  {
   p=malloc(c);
   add[x]=p;
   d[x]=c;
   printf("\n%c \t %d \t identifier\n",c,p);
   x++;
   j++;
  }
  else
  {
   ch=c;
   if(ch=='+'||ch=='-'||ch=='*'||ch=='=')
   {
    p=malloc(ch);
    add[x]=p;
    d[x]=ch;
    printf("\n %c \t %d \t operator\n",ch,p);
    x++;
    j++;
   }
}
}
}
```

OUTPUT:



```
l2sys29@l2sys29-Veriton-M275: ~/Desktop/syedvirus
l2sys29@l2sys29-Veriton-M275:~/Desktop/syedvirus$ ./exp1_symtab
Expression terminated by $:A+B+C=D$
Given Expression:A+B+C=D
 Symbol Table
Symbol    addr     type
A         25731088        identifier

 +        25731168        operator

B         25731232        identifier

 +        25731312        operator

C         25731376        identifier

 =        25731456        operator

D         25731536        identifier
l2sys29@l2sys29-Veriton-M275:~/Desktop/syedvirus$ 
```

Ex.No:04

Develop a lexical analyzer to recognize a few patterns in C.

(Ex. identifiers, constants, comments, operators etc.)

Program:

```
#include<stdio.h>

#include<conio.h>

#include<ctype.h>

#include<string.h> void main()

{

FILE *fi,*fo,*fop,*fk; int flag=0,i=1;

char c,t,a[15],ch[15],file[20]; clrscr();

printf("\n Enter the File Name:"); scanf("%s",&file);

fi=fopen(file,"r"); fo=fopen("inter.c","w");

fop=fopen("Oper.c","r");

fk=fopen("key.c","r"); c=getc(fi); while(!feof(fi))

{

if(isalpha(c)||isdigit(c)||(c=='['||c==']'||c=='.'==1)) fputc(c,fo);

else

{

if(c=='\n') fprintf(fo,"\t$\t");

else fprintf(fo,"\t%c\t",c);

}

c=getc(fi);

}

fclose(fi); fclose(fo);

fi=fopen("inter.c","r"); printf("\n Lexical Analysis"); fscanf(fi,"%s",a);

printf("\n Line: %d\n",i++); while(!feof(fi))

{

if(strcmp(a,"$")==0)

{

printf("\n Line: %d \n",i++); fscanf(fi,"%s",a);
}
fscanf(fop,"%s",ch);
while(!feof(fop))
{
if(strcmp(ch,a)==0)
{
fscanf(fop,"%s",ch); printf("\t\t%s\t:\t%s\n",a,ch); flag=1;
}
fscanf(fop,"%s",ch);
}
rewind(fop); fscanf(fk,"%s",ch);
while(!feof(fk))
{
if(strcmp(ch,a)==0)
{
fscanf(fk,"%k",ch); printf("\t\t%s\t:\tKeyword\n",a); flag=1;
}
fscanf(fk,"%s",ch);
}
rewind(fk); if(flag==0)
{
if(isdigit(a[0])) printf("\t\t%s\t:\tConstant\n",a);
else
printf("\t\t%s\t:\tIdentifier\n",a);
}
flag=0; fscanf(fi,"%s",a);
}
getch();
}
```

Key.C

int void main char if

for

while else printf scanf FILE

include stdio.h conio.h iostream.h

Oper.C

(open para

) closepara

{ openbrace

} closebrace

< lesser

> greater

" doublequote ' singlequote

: colon

; semicolon

# preprocessor

= equal

== asign

% percentage

^ bitwise

& reference

* star

+ add

- sub

\ backslash

/ slash

INPUT.C

```
#include "stdio.h"

#include "conio.h" void main()

{

int a=10,b,c; a=b*c; getch();

}
```

OUTPUT:

```
Line:1

# : preprocessor include : Identifier &quot; : doublequote stdio.h : Keyword &quot; :

doublequote

Line: 2

# : preprocessor include : Identifier &quot; : doublequote conio.h : Keyword &quot; :

doublequote

Line: 3

void : Keyword main : Keyword ( : open

) : closepara

Line: 4

{ : openbrace

Line: 5

int : Keyword a : Identifier

= : equal

10 : Constant

, : Identifier b : Identifier

, : Identifier c : Identifier

; : semicolon

Line: 6

a : Identifier

= : equal

b : Identifier

* : star
c : Identifier

; : semicolon

Line: 7

getch : Identifier ( : open

) : closepara

; : semicolon

Line: 8

} : clos
```

Ex.No:05

Program Which Prints Number Of Characters, Spaces, Tabs And Lines In A Text File

Program:

```
#include <stdio.h>
int main()
{
    char in_name[80];
    FILE *in_file;
    int ch, character = 0, line = 0, space = 0, tab = 0;
    printf("Enter file name:\n");
    scanf("%s", in_name);
     in_file = fopen(in_name, "r");
      if (in_file == NULL)
        printf("Can't open %s for reading.\n", in_name);
     else
    {
        while ((ch = fgetc(in_file)) != EOF)
        {
       character++;
            if (ch == ' ')
                space++;
            if (ch == '\n')
                line++;
            if (ch == '\t')
                tab++;
        }
        fclose(in_file);
        printf("\nNumber of characters = %d", character);
        printf("\nNumber of spaces = %d", space);
        printf("\nNumber of tabs = %d", tab);
printf("\nNumber of lines = %d", line);
   }
   return 0;
}
```

Count.txt

```
Hello,

This is line 1.
This is line 2.
This is line 3.
This is line 4.

              Thanks.
```

OUTPUT:

```
Enter file name:
count.txt

Number of characters = 82
Number of spaces = 12
Number of tabs = 1
Number of lines = 8
```

Ex.No:06

IMPLEMENTATION OF SYMBOL TABLE

Program:

LEX PART:

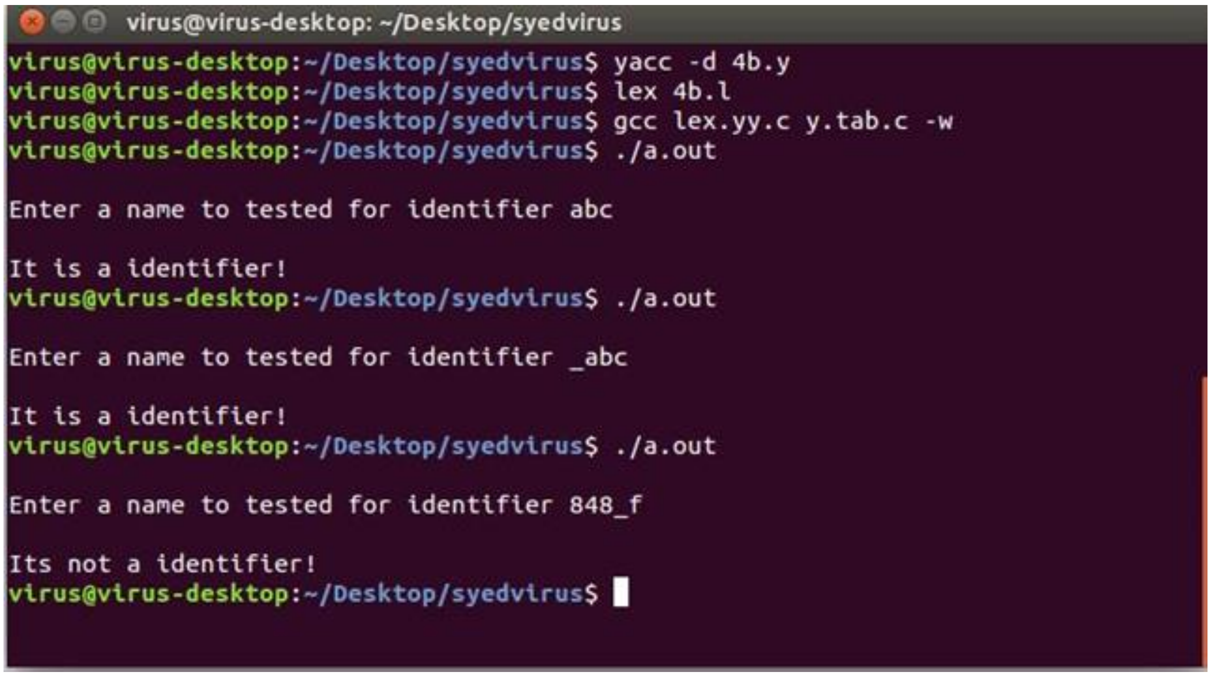
```
%{
 #include "y.tab.h"
%}
%%
[a-zA-Z_][a-zA-Z_0-9]* return letter;
[0-9]                  return digit;
.                return yytext[0];
\n                return 0;
%%
int yywrap()
{
return 1;
}
```

YACC PART:

```
%{
 #include<stdio.h>
int valid=1;
%}
%token digit letter
%%
start : letter s
s :    letter s
     | digit s
     |
     ;
%%
```

```
int yyerror()
{
   printf("\nIts not a identifier!\n");
    valid=0;
   return 0;
}
int main()
{
   printf("\nEnter a name to tested for identifier ");
   yyparse();
   if(valid)
   {
      printf("\nIt is a identifier!\n");
   }
}
```

OUTPUT:



```
virus@virus-desktop:~/Desktop/syedvirus$ yacc -d 4b.y
virus@virus-desktop:~/Desktop/syedvirus$ lex 4b.l
virus@virus-desktop:~/Desktop/syedvirus$ gcc lex.yy.c y.tab.c -w
virus@virus-desktop:~/Desktop/syedvirus$ ./a.out

Enter a name to tested for identifier abc

It is a identifier!
virus@virus-desktop:~/Desktop/syedvirus$ ./a.out

Enter a name to tested for identifier _abc

It is a identifier!
virus@virus-desktop:~/Desktop/syedvirus$ ./a.out

Enter a name to tested for identifier 848_f

Its not a identifier!
virus@virus-desktop:~/Desktop/syedvirus$ 
```

Ex.No:07

TO Implement Shift Reduce Parse

Program:

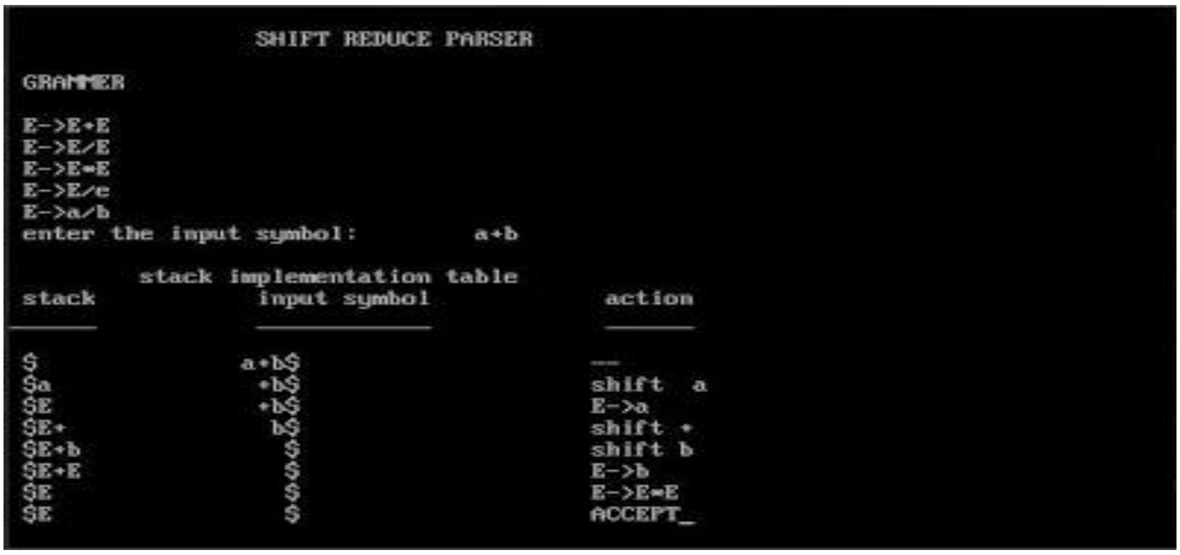
```
#include"stdio.h"

#include"stdlib.h"

#include"conio.h"

#include"string.h"

char ip_sym[15],stack[15];

int ip_ptr=0,st_ptr=0,len,i;

char temp[2],temp2[2];

char act[15];

void check();

void main(){

clrscr();

printf("\n\t\t SHIFT REDUCE PARSER\n");

printf("\n GRAMMER\n");

printf("\n E->E+E\n E->E/E");

printf("\n E->E*E\n E->a/b");

printf("\n enter the input symbol:\t");

gets(ip_sym);

printf("\n\t stack implementation table");

printf("\n stack\t\t input symbol\t\t action");

printf("\n______\t\t ____________\t\t ______\n");

printf("\n $\t\t%s$\t\t\t--",ip_sym);

strcpy(act,"shift ");

temp[0]=ip_sym[ip_ptr];
```

```
temp[1]='\0';

strcat(act,temp);

len=strlen(ip_sym);

for(i=0;i<=len-1;i++){

stack[st_ptr]=ip_sym[ip_ptr];

stack[st_ptr+1]='\0';

ip_sym[ip_ptr]=' ';

ip_ptr++;

printf("\n $%s\t\t%s$\t\t\t%s",stack,ip_sym,act);

strcpy(act,"shift ");

temp[0]=ip_sym[ip_ptr];

temp[1]='\0';

strcat(act,temp);

check();

st_ptr++;

}

st_ptr++;

check();

}

void check()

{

int flag=0;

temp2[0]=stack[st_ptr];

temp2[1]='\0';

if((!strcmpi(temp2,"a"))||(!strcmpi(temp2,"b")))
```

```
{
 stack[st_ptr]='E';
 if(!strcmpi(temp2,"a"))
  printf("\n $%s\t\t%s$\t\t\tE->a",stack, ip_sym);
 else
  printf("\n $%s\t\t%s$\t\t\tE->b",stack,ip_sym);
 flag=1;
}
if((!strcmpi(temp2,"+"))||(strcmpi(temp2,"*"))||(!strcmpi(temp2,"/")))
{
 flag=1;
}
if((!strcmpi(stack,"E+E"))||(!strcmpi(stack,"E\E"))||(!strcmpi(stack,"E*E")))
{
strcpy(stack,"E");
st_ptr=0;
if(!strcmpi(stack,"E+E"))
printf("\n $%s\t\t%s$\t\t\tE->E+E",stack,ip_sym);
else
if(!strcmpi(stack,"E\E"))
printf("\n $%s\t\t %s$\t\t\tE->E\E",stack,ip_sym);
else
printf("\n $%s\t\t%s$\t\t\tE->E*E",stack,ip_sym);
flag=1;
}
```

```
if(!strcmpi(stack,"E")&&ip_ptr==len)

{

printf("\n $%s\t\t%s$\t\t\tACCEPT",stack,ip_sym);

getch();

exit(0);

}

if(flag==0)

{

printf("\n%s\t\t\t%s\t\t reject",stack,ip_sym);

exit(0);

}

return;

}
```

OUTPUT:



```
                  SHIFT REDUCE PARSER

 GRAMMER

 E->E+E
 E->E/E
 E->E*E
 E->E/e
 E->a/b
 enter the input symbol:        a+b

         stack implementation table
 stack             input symbol            action
______             ____________            ______

 $                a+b$                     --
 $a                +b$                    shift  a
 $E                +b$                    E->a
 $E+                b$                    shift +
 $E+b                $                    shift b
 $E+E                $                    E->b
 $E                  $                    E->E*E
 $E                  $                    ACCEPT_
```

Ex.No:08

Construction of LR Parsing table

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<string.h>
char stack[30];
int top=-1;
void push(char c)
{
top++;
stack[top]=c;
}
char pop()
{
char c;
if(top!=-1)
{
c=stack[top];
top--;
return c;
}
return'x';
}
void printstat()
{
int i;
printf("\n\t\t\t $");
for(i=0;i<=top;i++)
printf("%c",stack[i]);
```

```
}
void main()
{
int i,j,k,l;
char s1[20],s2[20],ch1,ch2,ch3;
clrscr();
printf("\n\n\t\t LR PARSING");
printf("\n\t\t ENTER THE EXPRESSION");
scanf("%s",s1);
l=strlen(s1);
j=0;
printf("\n\t\t $");
for(i=0;i
{
if(s1[i]=='i' && s1[i+1]=='d')
{
s1[i]=' ';
s1[i+1]='E';
printstat(); printf("id");
push('E');
printstat();
}
else if(s1[i]=='+'||s1[i]=='-'||s1[i]=='*' ||s1[i]=='/' ||s1[i]=='d')
{
push(s1[i]);
printstat();
}
}
printstat();
l=strlen(s2);
while(l)
{
ch1=pop();
if(ch1=='x')
{
printf("\n\t\t\t $");
break;
}
if(ch1=='+'||ch1=='/'||ch1=='*'||ch1=='-')
{
ch3=pop();
if(ch3!='E')
{
printf("errror");
exit();
}
else
{
push('E');
printstat();
}
}
ch2=ch1;
}
getch();
}
```

OUTPUT:

```
LR PARSING

ENTER THE EXPRESSION

id+id*id-id

$

$id

$E

$E+

$E+id

$E+E

$E+E*

$E+E*id

$E+E*E

$E+E*E-

$E+E*E-id

$E+E*E-E

$E+E*E-E

$E+E*E

$E

$
```

Ex.No:09

IMPLEMENTATION OF CALCULATOR USING Lex & YACC

Program:

LEX PART:

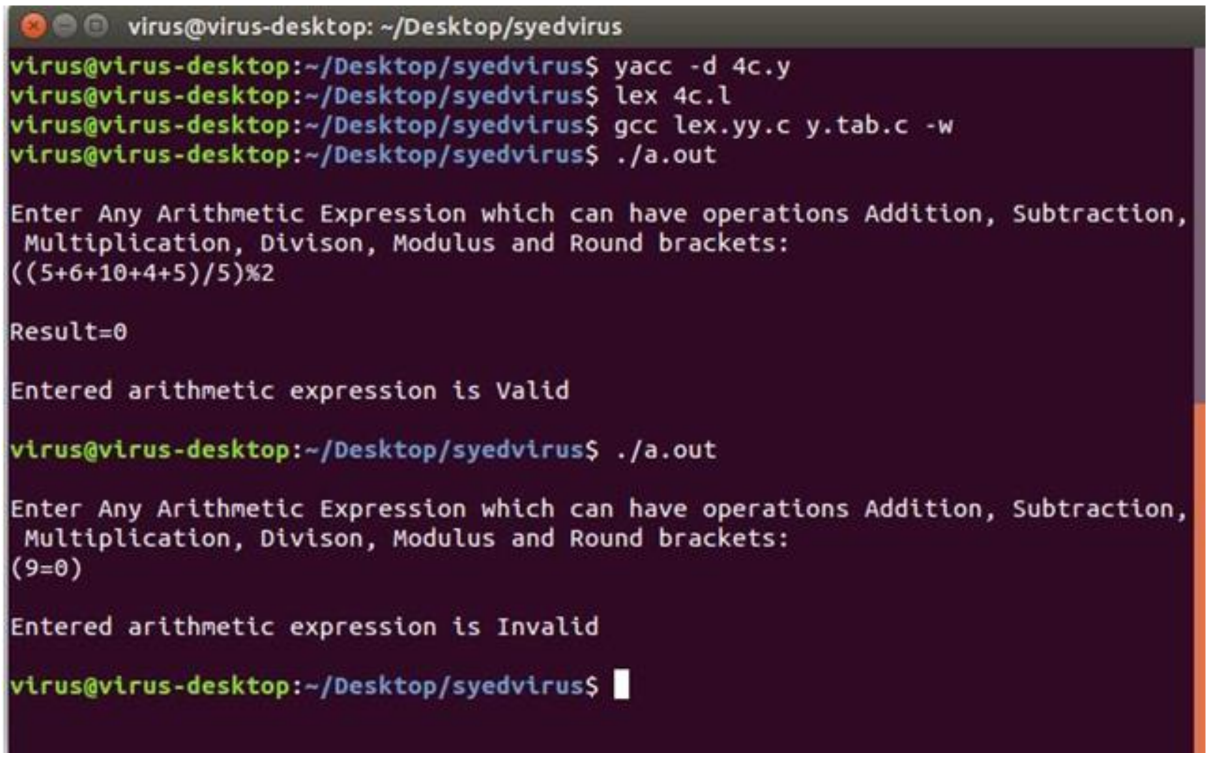
```
%{
#include<stdio.h>
#include "y.tab.h"
extern int yylval;
%}
%%
[0-9]+ {
          yylval=atoi(yytext);
          return NUMBER;
       }
[\t] ;
[\n] return 0;
. return yytext[0];
%%
int yywrap()
{
return 1;
}
```

YACC PART:

```
%{
   #include<stdio.h>
   int flag=0;
%}
%token NUMBER
%left '+' '-'
%left '*' '/' '%'
```

```
%left '(' ')'

%%

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;}

;

%%

void main()

{

   printf("\nEnter Any Arithmetic Expression which can have operations Addition, Subtraction,
Multiplication, Divison, 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:



```
virus@virus-desktop:~/Desktop/syedvirus$ yacc -d 4c.y
virus@virus-desktop:~/Desktop/syedvirus$ lex 4c.l
virus@virus-desktop:~/Desktop/syedvirus$ gcc lex.yy.c y.tab.c -w
virus@virus-desktop:~/Desktop/syedvirus$ ./a.out

Enter Any Arithmetic Expression which can have operations Addition, Subtraction,
 Multiplication, Divison, Modulus and Round brackets:
((5+6+10+4+5)/5)%2

Result=0

Entered arithmetic expression is Valid

virus@virus-desktop:~/Desktop/syedvirus$ ./a.out

Enter Any Arithmetic Expression which can have operations Addition, Subtraction,
 Multiplication, Divison, Modulus and Round brackets:
(9=0)

Entered arithmetic expression is Invalid

virus@virus-desktop:~/Desktop/syedvirus$ 
```

Ex.No:10

To Recognize A Valid Arithmetic Expression

Program:

LEX PART:

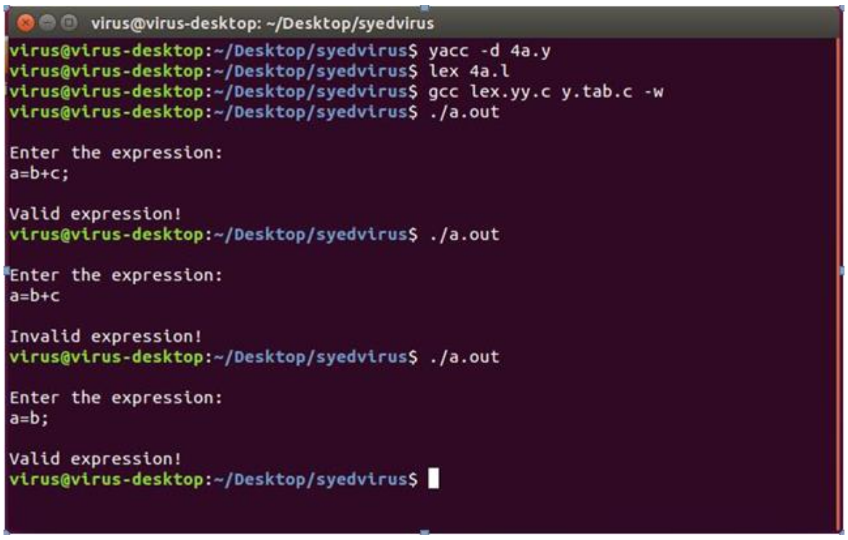
```
%{
   #include "y.tab.h"
%}
%%
[a-zA-Z_][a-zA-Z_0-9]* return id;
[0-9]+(\.[0-9]*)?      return num;
[+/*]                return op;
.                    return yytext[0];
\n                   return 0;
%%
int yywrap()
{
return 1;
}
```

YACC PART:

```
%{
   #include<stdio.h>
   int valid=1;
%}
%token num id op
%%
start : id '=' s ';'
s :    id x
     | num x
     | '-' num x
     | '(' s ')' x
```

```
      ;
x :     op s
      | '-' s
      |
      ;
%%
int yyerror()
{
   valid=0;
   printf("\nInvalid expression!\n");
   return 0;
}
int main()
{
   printf("\nEnter the expression:\n");
   yyparse();
   if(valid)
   {
      printf("\nValid expression!\n");
   }
}
```

OUTPUT:



```
virus@virus-desktop:~/Desktop/syedvirus$ yacc -d 4a.y
virus@virus-desktop:~/Desktop/syedvirus$ lex 4a.l
virus@virus-desktop:~/Desktop/syedvirus$ gcc lex.yy.c y.tab.c -w
virus@virus-desktop:~/Desktop/syedvirus$ ./a.out

Enter the expression:
a=b+c;

Valid expression!
virus@virus-desktop:~/Desktop/syedvirus$ ./a.out

Enter the expression:
a=b+c

Invalid expression!
virus@virus-desktop:~/Desktop/syedvirus$ ./a.out

Enter the expression:
a=b;

Valid expression!
virus@virus-desktop:~/Desktop/syedvirus$ 
```

Ex.No:11

To Implement Syntax Tree

Program:

```
#include<conio.h>

#include<stdio.h>

void main()

{

FILE *fp;

int i=0,j=0,k,l,row,col,s,x;

char a[10][10],ch,main[50],search;

clrscr();

fp=fopen("syntax.txt","r+");

while((ch=fgetc(fp))!=EOF)

{

if(ch=='\n')

   {

   row=i;

   col=j;

   j=0;

   i++;

   }

else

   {

   a[i][j]=ch;

   j++;

   }

}

printf("\n");

for(k=0;k<row+1;k++)

{

for(l=0;l<col;l++)
```

```
{
printf("%c",a[k][l]);
}
printf("\n");
}
i=0;
s=0;
for(k=0;k<row+1;k++)
{
    main[i]=a[k][1];
    i++;
    if(a[k][3]=='t')
    {
      search=a[k][4];
      for(l=0;l<i;l++)
      {
      if(main[l]==search)
        {
        main[i]=main[l];
        i++;
        break;
        }
      }
    main[i]=a[k][5];
    s=5;
    i++;
    }
    else
    {
      main[i]=a[k][3];
         //   printf("\n%c",main[i]);
```

```
            i++;
            main[i]=a[k][4];
              //    printf(",%c\n",main[i]);
            s=4;
            i++;
        }
        s++;
        if(a[k][s]=='t')
        {
        s++;
        search=a[k][s];
        for(l=0;l<i;l++)
            {
            if(main[l]==search)
                {
                main[i]=main[l];
                i++;
                break;
                }
            }
        }
        else
        {
        main[i]=a[k][s];
        i++;
        }
    }
        for(x=i-1;x>=0;x=x-4)
        {
        printf("\ntt%c: root->%c ",main[x-3],main[x-1]);
        if(main[x-2]>48 &&main[x-2]<59)
            printf("lc->t%c ",main[x-2]);
        else
            printf("lc->%c ",main[x-2]);
        if(main[x]>48 &&main[x]<59)
            printf("rc->t%c ",main[x]);
        else
            printf("rc->%c ",main[x]);
        }
    getch();
    }
```

Syntax.txt

```
t1=a+b
```

OUTPUT:

```
t1=a+b
tt1:root->+     lc->a      rc->b
```

Ex.No:12

Three address code generation for assignment statement

Program:

```
#include<stdio.h>
char s[20],t[20];
void main()
{
printf("\nEnter expression:");
scanf("%s",&t);
printf("\nIntermediate code is:");
if(isalpha(t[2])&&isalpha(t[0])&&isalpha(t[4]))
{
printf("\n mov%c.r",t[2]);
else
printf("\nEnter correct expression!");switch(t[3])
{
case '*':
printf("\n mul %c.r".t[4]);
printf("\n mov r.%c",t[0]); break;
case '+':
printf("\n add %c.r",t[4]);
printf("\n mov r.%c",t[0]); break;
case '-':
printf("\n sub %c.r",t[4]);
printf("\n mov r.%c",t[0]); break;
case '/':
printf("\n div %c.r",t[4]);
printf("\n mov r.%c",t[0]); break;
default:
printf("\nInvalid expression!"); break;
} }
```

Output:

```
./a.out

Enter expression:a=a+b

Intermediate code is:

mova.r

add b.r
```