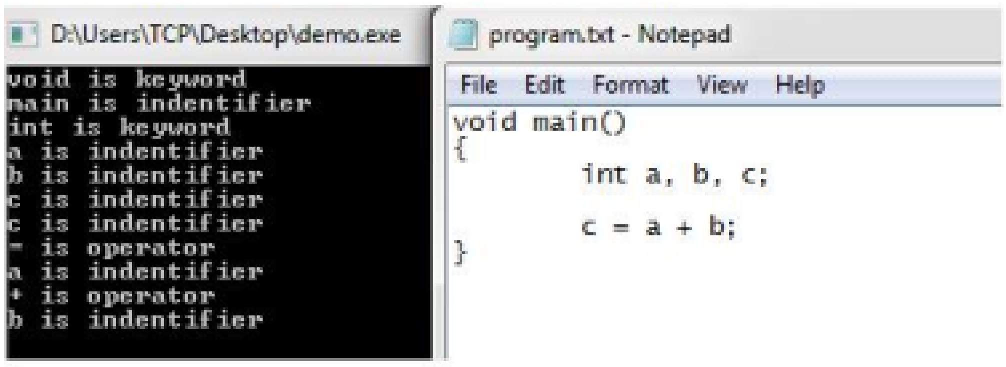
Experiment - 6: **Implementation of Lexical Analyzer using C.**

Lexical.c

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

int isKeyword(char buffer[]){
char keywords[32][10] = {"auto","break","case","char","const","continue","default",
"do","double","else","enum","extern","float","for","goto",
"if","int","long","register","return","short","signed",
"sizeof","static","struct","switch","typedef","union",
"unsigned","void","volatile","while"};
int i, flag = 0;

for(i = 0; i < 32; ++i){
if(strcmp(keywords[i], buffer) == 0){
flag = 1;
break;
}
}

return flag;
}

int main(){
char ch, buffer[15], operators[] = "+-*/%=";
FILE *fp;
int i,j=0;

fp = fopen("program.txt","r");

if(fp == NULL){
printf("error while opening the file\n");
exit(0);
}

while((ch = fgetc(fp)) != EOF){
for(i = 0; i < 6; ++i){
if(ch == operators[i])
printf("%c is operator\n", ch);
}

if(isalnum(ch)){
buffer[j++] = ch;
}
else if((ch == ' ' || ch == '\n') && (j != 0)){
buffer[j] = '\0';
               j = 0;

                if(isKeyword(buffer) == 1)
                   printf("%s is keyword\n", buffer);
                else
              printf("%s is indentifier\n", buffer);
          }

    }

    fclose(fp);

   return 0;
}
```

OUTPUT



```
void is keyword
main is indentifier
int is keyword
a is indentifier
b is indentifier
c is indentifier
c is indentifier
= is operator
a is indentifier
+ is operator
b is indentifier
```

```
void main()
{
        int a, b, c;

        c = a + b;
}
```

Experiment-7: Implementation of Lexical Analyzer using LEX tools.

Lexical.l

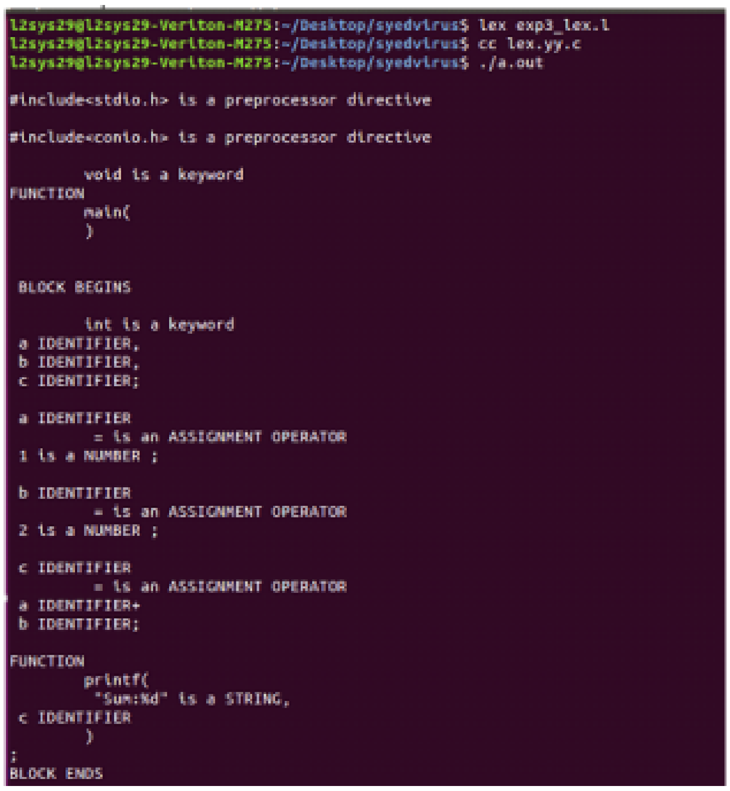
```
%{
int COMMENT=0;
%}
identifier [a-zA-Z][a-zA-Z0-9]*
%%
#.* {printf("\n%s is a preprocessor directive",yytext);}

int |float |char |double |while |for |struct |typedef |do |if |break |continue |void |switch |return |else
|goto  printf("\n\t%s is a keyword",yytext);}

"/*" {COMMENT=1;}{printf("\n\t %s is a COMMENT",yytext);}
{identifier}\( {if(!COMMENT)printf("\nFUNCTION \n\t%s",yytext);} \{
{if(!COMMENT)printf("\n BLOCK BEGINS");}
\} {if(!COMMENT)printf("BLOCK ENDS ");}
 {identifier}(\[[0-9]*\])? {if(!COMMENT) printf("\n %s IDENTIFIER",yytext);}
\".*\" {if(!COMMENT)printf("\n\t %s is a STRING",yytext);} [0-9]+
{if(!COMMENT) printf("\n %s is a NUMBER ",yytext);} \)(\:)?
{if(!COMMENT)printf("\n\t");ECHO;printf("\n");} \( ECHO;
= {if(!COMMENT)printf("\n\t %s is an ASSIGNMENT OPERATOR",yytext);} \<=
|\>= |\< |== |\> {if(!COMMENT) printf("\n\t%s is a RELATIONAL
OPERATOR",yytext);}%%

int main(int argc, char **argv)
 {
FILE *file;
 file=fopen("var.c","r");
if(!file)
 {
printf("could not open the file");
 exit(0);
}
yyin=file;
yylex();
printf("\n");
return(0);
}
int yywrap()
 {
return(1);
}
```

var.c

```
#include<stdio.h>
#include<conio.h>
void main()
```

```
{
int a,b,c;
a=1;
b=2;
c=a+b;
printf("Sum:%d",c);
}
```

OUTPUT:



```
l2sys29@l2sys29-Veriton-M275:~/Desktop/syedvirus$ lex exp3_lex.l
l2sys29@l2sys29-Veriton-M275:~/Desktop/syedvirus$ cc lex.yy.c
l2sys29@l2sys29-Veriton-M275:~/Desktop/syedvirus$ ./a.out

#include<stdio.h> is a preprocessor directive

#include<conio.h> is a preprocessor directive

        void is a keyword
FUNCTION
        main(
        )


 BLOCK BEGINS

        int is a keyword
 a IDENTIFIER,
 b IDENTIFIER,
 c IDENTIFIER;

 a IDENTIFIER
         = is an ASSIGNMENT OPERATOR
 1 is a NUMBER ;

 b IDENTIFIER
         = is an ASSIGNMENT OPERATOR
 2 is a NUMBER ;

 c IDENTIFIER
         = is an ASSIGNMENT OPERATOR
 a IDENTIFIER+
 b IDENTIFIER;

FUNCTION
        printf(
         "Sum:%d" is a STRING,
 c IDENTIFIER
        )
;
BLOCK ENDS
```

Experiment-8: LEX and YACC program to check valid variable followed by letter or digits.

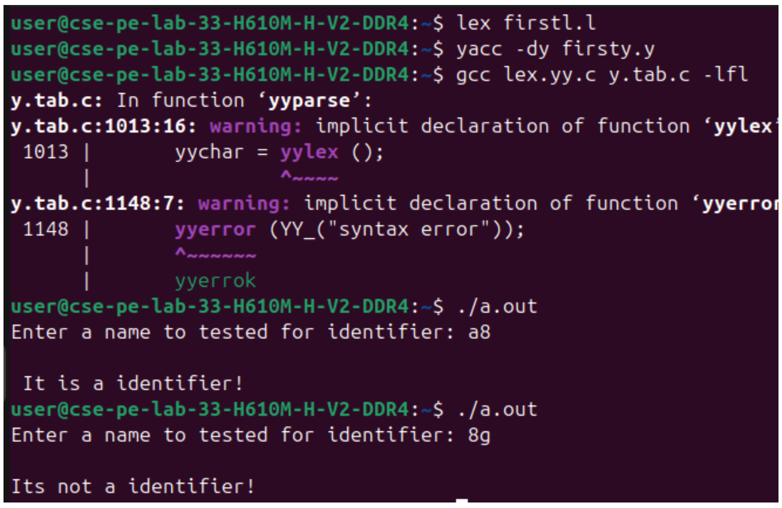
LEX CODE:

```
%{
 #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 CODE:

```
%{
 #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("\n It is a identifier!\n");
 }
}
```

OUTPUT



```
user@cse-pe-lab-33-H610M-H-V2-DDR4:~$ lex firstl.l
user@cse-pe-lab-33-H610M-H-V2-DDR4:~$ yacc -dy firsty.y
user@cse-pe-lab-33-H610M-H-V2-DDR4:~$ gcc lex.yy.c y.tab.c -lfl
y.tab.c: In function ‘yyparse’:
y.tab.c:1013:16: warning: implicit declaration of function ‘yylex’
 1013 |       yychar = yylex ();
      |                ^~~~~
y.tab.c:1148:7: warning: implicit declaration of function ‘yyerror
 1148 |       yyerror (YY_("syntax error"));
      |       ^~~~~~~
      |       yyerrok
user@cse-pe-lab-33-H610M-H-V2-DDR4:~$ ./a.out
Enter a name to tested for identifier: a8

 It is a identifier!
user@cse-pe-lab-33-H610M-H-V2-DDR4:~$ ./a.out
Enter a name to tested for identifier: 8g

Its not a identifier!
```

Experiment-9: LEX and YACC program to valid arithmetic expression.

LEX CODE:

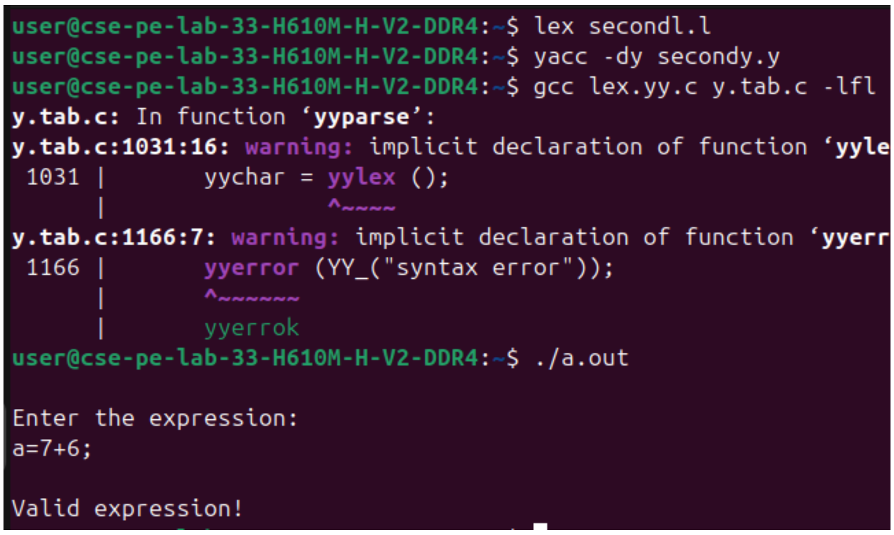
```
%{
#include "y.tab.h"
%}
%%
[a-zA-Z_][a-zA-Z_0-9]* return id;
[0-9]+(\.[0-9]*)? return num;
[+/*] return op;
. return yytext[0];
\n return 0;
%%
int yywrap()
{
return 1;
}
```

YACC CODE:

```
%{
#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



```
user@cse-pe-lab-33-H610M-H-V2-DDR4:~$ lex secondl.l
user@cse-pe-lab-33-H610M-H-V2-DDR4:~$ yacc -dy secondy.y
user@cse-pe-lab-33-H610M-H-V2-DDR4:~$ gcc lex.yy.c y.tab.c -lfl
y.tab.c: In function ‘yyparse’:
y.tab.c:1031:16: warning: implicit declaration of function ‘yyle
 1031 |       yychar = yylex ();
      |                ^~~~~
y.tab.c:1166:7: warning: implicit declaration of function ‘yyerr
 1166 |       yyerror (YY_("syntax error"));
      |       ^~~~~~~
      |       yyerrok
user@cse-pe-lab-33-H610M-H-V2-DDR4:~$ ./a.out

Enter the expression:
a=7+6;

Valid expression!
```

Experiment-10: Implement calculator using Lex and Yacc.

LEX CODE:

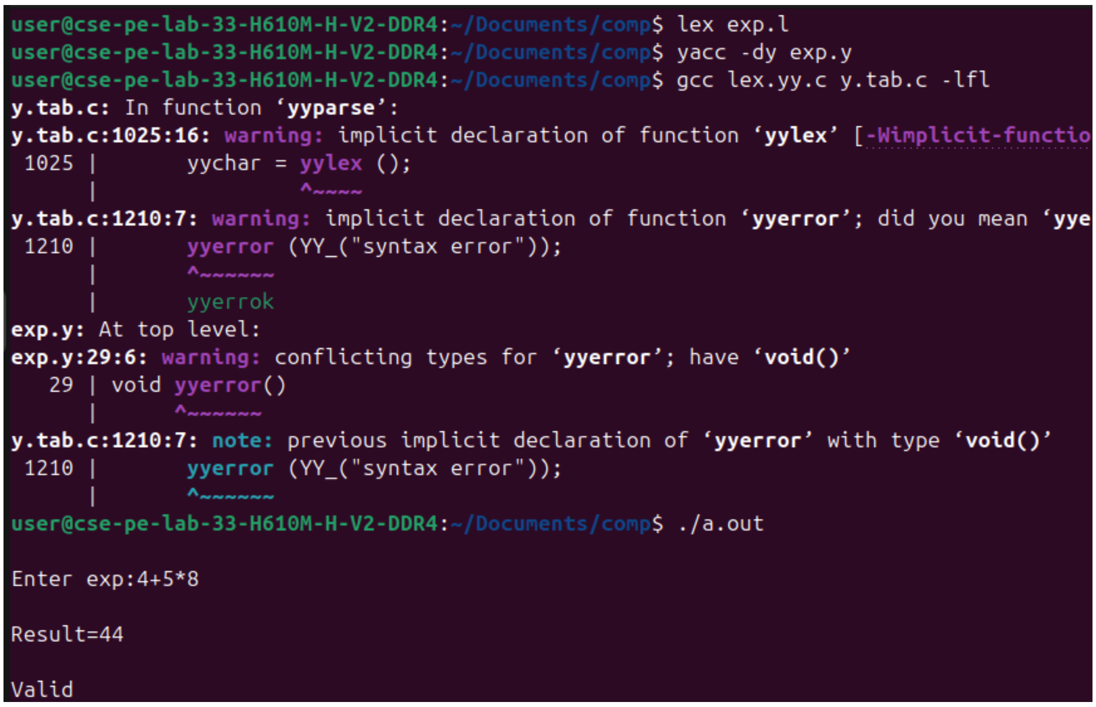
```
%{
#include<stdio.h>
#include "y.tab.h"
extern int yylval;
%}

%%
[0-9]+ {yylval = atoi(yytext); return NUMBER;}
[\t] ;
[\n] return 0;
. return yytext[0];
%%

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

YACC CODE:

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

%token NUMBER
%left '+' '-'
%left '*' '/' '%'
%left '(' ')'

%%
ArithmeticExpression: E {
   printf("Result:%d",$$);
   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("Enter the expression:\n");
    yyparse();
    if(flag == 0)                  // If no error, indicate the expression is valid
        printf("\nEntered arithmetic expression is Valid\n\n");
    return 0;
}

void yyerror(){
flag = 1;
printf("Invalid Input");

}
```

OUTPUT:

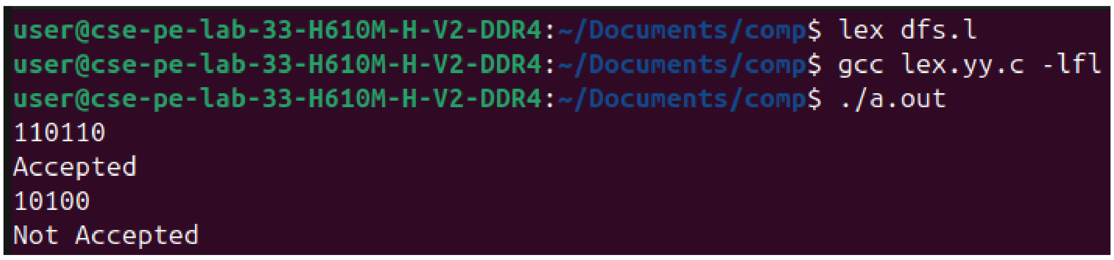


```
user@cse-pe-lab-33-H610M-H-V2-DDR4:~/Documents/comp$ lex exp.l
user@cse-pe-lab-33-H610M-H-V2-DDR4:~/Documents/comp$ yacc -dy exp.y
user@cse-pe-lab-33-H610M-H-V2-DDR4:~/Documents/comp$ gcc lex.yy.c y.tab.c -lfl
y.tab.c: In function ‘yyparse’:
y.tab.c:1025:16: warning: implicit declaration of function ‘yylex’ [-Wimplicit-functio
 1025 |       yychar = yylex ();
      |                ^~~~~
y.tab.c:1210:7: warning: implicit declaration of function ‘yyerror’; did you mean ‘yye
 1210 |       yyerror (YY_("syntax error"));
      |       ^~~~~~~
      |       yyerrok
exp.y: At top level:
exp.y:29:6: warning: conflicting types for ‘yyerror’; have ‘void()’
   29 | void yyerror()
      |      ^~~~~~~
y.tab.c:1210:7: note: previous implicit declaration of ‘yyerror’ with type ‘void()’
 1210 |       yyerror (YY_("syntax error"));
      |       ^~~~~~~
user@cse-pe-lab-33-H610M-H-V2-DDR4:~/Documents/comp$ ./a.out

Enter exp:4+5*8

Result=44

Valid
```

Experiment-11: Implement LEX program to accept Even number of ones and even number of zeros or Odd number of ones and even number of zeros.

```
%{
%}
%s A B
%%
<INITIAL>1 BEGIN INITIAL;
<INITIAL>0 BEGIN A;
<INITIAL>[^0|\n] BEGIN B;
<INITIAL>\n BEGIN INITIAL; printf("Accepted\n");
<A>1 BEGIN A;
<A>0 BEGIN INITIAL;
<A>[^0|\n] BEGIN B;
<A>\n BEGIN INITIAL; printf("Not Accepted\n");
<B>0 BEGIN B;
<B>1 BEGIN B;
<B>[^0|\n] BEGIN B;
<B>\n {BEGIN INITIAL; printf("INVALID\n");}
%%
void main()
{
yylex();
}
```

OUTPUT:

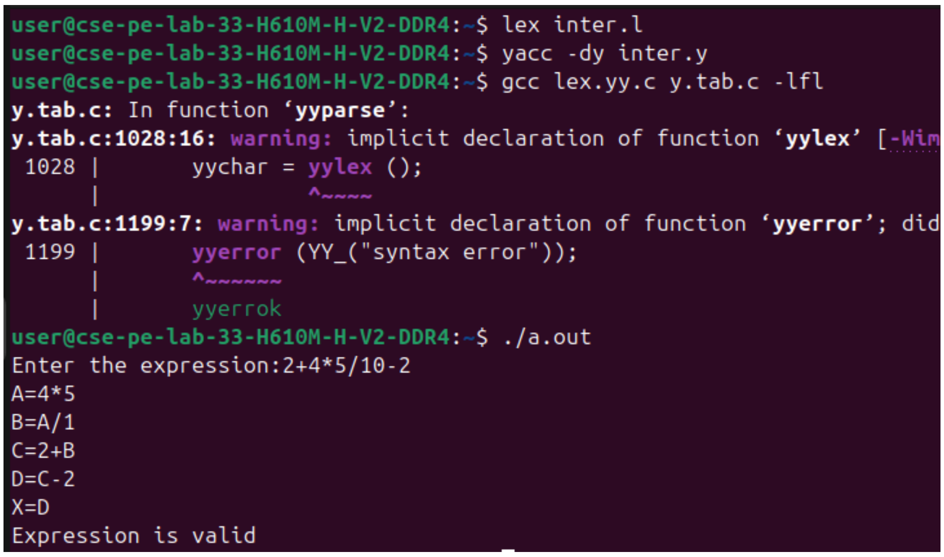


```
user@cse-pe-lab-33-H610M-H-V2-DDR4:~/Documents/comp$ lex dfs.l
user@cse-pe-lab-33-H610M-H-V2-DDR4:~/Documents/comp$ gcc lex.yy.c -lfl
user@cse-pe-lab-33-H610M-H-V2-DDR4:~/Documents/comp$ ./a.out
110110
Accepted
10100
Not Accepted
```

Experiment-12: Implement Intermediate Code Generation for simple expressions using LEX.

```
%{
        #include"y.tab.h"
%}
%%
[0-9]+ {yylval.dval=(*yytext);return NUM;} \n return 0;
. return yytext[0];
%%
%{
        #include<stdio.h>
        #include<stdlib.h>
        char p='A';
%}
%union {char dval;}
%token NUM sign
%left '+''-'
%left '*''/'
%type <dval> S
%type <dval> E
%%
S:E {printf("X=%c\n",$$);} E:NUM {}|E'+'E
{printf("%c=%c+%c\n",p,$1,$3);$$=p;p++;}
        |E'-'E {printf("%c=%c- %c\n",p,$1,$3);$$=p;p++;} |E'*'E
        {printf("%c=%c*%c\n",p,$1,$3);$$=p;p ++;} |E'/'E
        {printf("%c=%c/%c\n",p,$1,$3);$$=p;p ++;}
%%
int main()
{
        printf("Enter the expression : ");
        yyparse();
        printf("Expression is valid\n");
        return 0;
}
int yyerror()
{
        printf("Expression is invalid \n");
        exit(0);
}
```

OUTPUT:

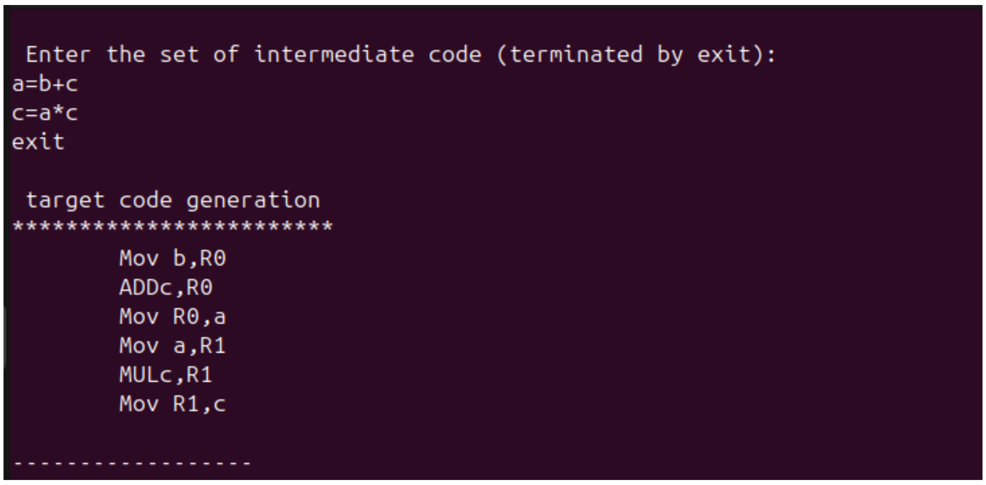


```
user@cse-pe-lab-33-H610M-H-V2-DDR4:~$ lex inter.l
user@cse-pe-lab-33-H610M-H-V2-DDR4:~$ yacc -dy inter.y
user@cse-pe-lab-33-H610M-H-V2-DDR4:~$ gcc lex.yy.c y.tab.c -lfl
y.tab.c: In function ‘yyparse’:
y.tab.c:1028:16: warning: implicit declaration of function ‘yylex’ [-Wim
 1028 |       yychar = yylex ();
      |                ^~~~~
y.tab.c:1199:7: warning: implicit declaration of function ‘yyerror’; did
 1199 |       yyerror (YY_("syntax error"));
      |       ^~~~~~~
      |       yyerrok
user@cse-pe-lab-33-H610M-H-V2-DDR4:~$ ./a.out
Enter the expression:2+4*5/10-2
A=4*5
B=A/1
C=2+B
D=C-2
X=D
Expression is valid
```

Experiment-13: To Implement the back end of the compiler which takes the three address code and produces the 8086 assembly language instructions that can be assembled and run using an 8086 assembler. The target assembly instructions can be simple move, add, sub, jump etc.

```
#include<stdio.h>
#include<stdio.h>
//#include<conio.h>
#include<string.h>
void main()
{
 char icode[10][30],str[20],opr[10];
int i=0;
//clrscr();
printf("\n Enter the set of intermediate code (terminated by exit):\n");
do
{
 scanf("%s",icode[i]);
} while(strcmp(icode[i++],"exit")!=0);
printf("\n target code generation");
printf("\n************************");
i=0;
do
{
 strcpy(str,icode[i]);
 switch(str[3])
 {
 case '+':
 strcpy(opr,"ADD");
 break;
 case '-':
 strcpy(opr,"SUB");
 break;
 case '*':
 strcpy(opr,"MUL");
 break;
 case '/':
 strcpy(opr,"DIV");
 break;
 }
 printf("\n\tMov %c,R%d",str[2],i);

 printf("\n\t%s%c,R%d",opr,str[4],i);
 printf("\n\tMov R%d,%c",i,str[0]);
 }while(strcmp(icode[++i],"exit")!=0);
//getch();
}
```

OUTPUT:

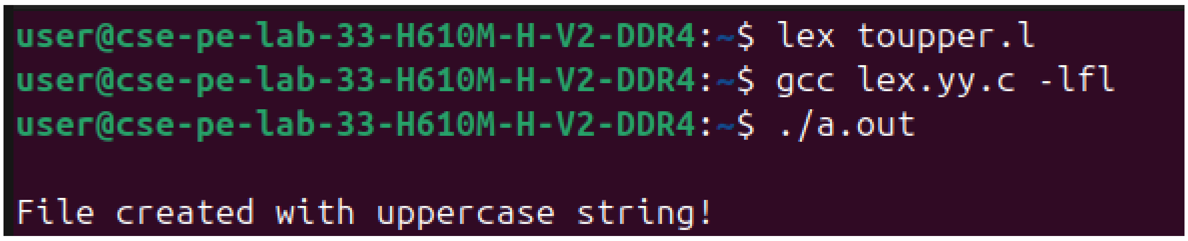


```
 Enter the set of intermediate code (terminated by exit):
a=b+c
c=a*c
exit

 target code generation
************************
        Mov b,R0
        ADDc,R0
        Mov R0,a
        Mov a,R1
        MULc,R1
        Mov R1,c

------------------
```

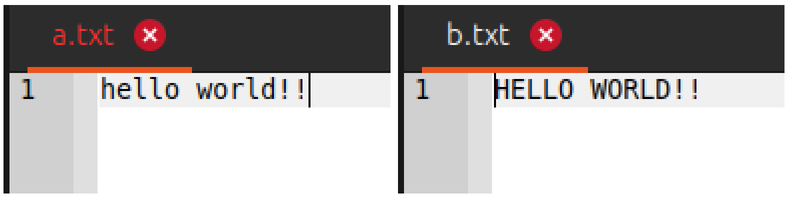
Experiment-14: LEX Program to convert the substring abc to ABC from the given input string.

```
%{
        FILE *fp;
        FILE *fp1;
%}
%%
[a-z] fprintf(fp1,"%c",toupper(yytext[0]));
. fprintf(fp1,"%C",yytext[0]);
%%
int main(int args,char **argv)
{
        fp1=fopen("b.txt","w");
        if(!fp1)
        {
                printf("file not exists\n");
                return(0);
        }
        fp=fopen("a.txt","r");
        if(!fp)
        {
                printf("file not exists\n");
                return(0);
        }
        yyin=fp1;
        yyin=fp;
        yylex();
        return(0);
}
```

OUTPUT:



```
user@cse-pe-lab-33-H610M-H-V2-DDR4:~$ lex toupper.l
user@cse-pe-lab-33-H610M-H-V2-DDR4:~$ gcc lex.yy.c -lfl
user@cse-pe-lab-33-H610M-H-V2-DDR4:~$ ./a.out

File created with uppercase string!
```



a.txt

```
hello world!!
```

b.txt

```
HELLO WORLD!!
```