1. Design and implement a lexical analyzer using C language to recognize all valid tokens in the input program. The lexical analyzer should ignore redundant spaces, tabs and newlines. It should also ignore comments.

AIM: Design a lexical analyzer for given language and the lexical analyzer should ignore redundant spaces, tabs and new lines. It should also ignore comments. Although the syntax specification states that identifiers can be arbitrarily long, you may restrict the length to some reasonable value. Simulate the same in C language.

LOGIC:

1. Read the input Expression

2. Check whether input is alphabet or digits then store it as identifier

3. If the input is is operator store it as symbol

4. Check the input for keywords

PROGRAM:

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>
void keyword(char str[10])
{
if(strcmp("for",str)==0||strcmp("while",str)==0||strcmp("do",str)==0||
strcmp("int",str)==0||strcmp("float",str)==0||strcmp("char",str)==0||
strcmp("double",str)==0||strcmp("static",str)==0||
strcmp("switch",str)==0||strcmp("case",str)==0)
printf("\n%s is a keyword",str);
else
printf("\n%s is an identifier",str);
}

main()
{
FILE *f1,*f2,*f3;
char c,str[10],st1[10];
int num[100],lineno=0,tokenvalue=0,i=0,j=0,k=0;
printf("\nEnter the c program");/*gets(st1);*/
f1=fopen("input","w");
while((c=getchar())!=EOF)
putc(c,f1);
fclose(f1);
f1=fopen("input","r");
f2=fopen("identifier","w");
f3=fopen("specialchar","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=='\t')
printf(" ");
else
if(c=='\n')
lineno++;
else
putc(c,f3);
}
fclose(f2);
fclose(f3);
fclose(f1);
printf("\nThe no's in the program are");
for(j=0;j
printf("%d",num[j]);
printf("\n");
f2=fopen("identifier","r");
k=0;
printf("The keywords and identifiersare:");
while((c=getc(f2))!=EOF){
if(c!=' ')
str[k++]=c;
else
{
str[k]='\0';
keyword(str);
k=0;
}
}
fclose(f2);
f3=fopen("specialchar","r");
printf("\nSpecial characters are");
while((c=getc(f3))!=EOF)
printf("%c",c);
printf("\n");
fclose(f3);
printf("Total no. of lines are:%d",lineno);
}
```

Input:

```
Enter Program $ for termination:
{
int a[3],t1,t2;
t1=2; a[0]=1; a[1]=2; a[t1]=3;
t2=-(a[2]+t1*6)/(a[2]-t1);
if t2>5 then
print(t2);
else {
int t3;
t3=99;
t2=-25;
print(-t1+t2*t3); /* this is a comment on 2 lines */
} endif
}
(cntrl+z)
```

Output:

```
Variables : a[3] t1 t2 t3
Operator : - + * / >
Constants : 2 1 3 6 5 99 -25
Keywords : int if then else endif
Special Symbols : , ; ( ) { }
Comments : this is a comment on 2 lines
```

2. Implement a Lexical Analyzer for a given program using Lex Tool.

AIM:

To write a program for implementing a Lexical analyser using LEX tool in Linux platform.

ALGORITHM:

Step1: Lex program contains three sections: definitions, rules, and user subroutines. Each section must be separated from the others by a line containing only the delimiter, %%. The format is as follows: definitions %% rules %% user_subroutines

Step2: In definition section, the variables make up the left column, and their definitions make up the right column. Any C statements should be enclosed in %{..}%. Identifier is defined such that the first letter of an identifier is alphabet and remaining letters are alphanumeric.

Step3: In rules section, the left column contains the pattern to be recognized in an input file to yylex(). The right column contains the C program fragment executed when that pattern is recognized. The various patterns are keywords, operators, new line character, number, string, identifier, beginning and end of block, comment statements, preprocessor directive statements etc.

Step4: Each pattern may have a corresponding action, that is, a fragment of C source code to execute when the pattern is matched.

Step5: When yylex() matches a string in the input stream, it copies the matched text to an external character array, yytext, before it executes any actions in the rules section.

Step6: In user subroutine section, main routine calls yylex(). yywrap() is used to get more input.

Step7: The lex command uses the rules and actions contained in file to generate a program, lex.yy.c, which can be compiled with the cc command. That program can then receive input, break the input into the logical pieces defined by the rules in file, and run program fragments contained in the actions in file.

PROGRAM CODE:

```
/* program name is lexp.l */
%{
/* program to recognize a c program */
int COMMENT=0;
int cnt=0;
%}
identifier [a-zA-Z][a-zA-Z0-9]*
%%
#.* { printf("\n%s is a PREPROCESSOR DIRECTIVE",yytext);}
int |
float |
char |
double |
while |
for |
do |
if |
break |
continue |
void |
switch |
case |
long |
struct |
const |
typedef |
return |
else |
goto {printf("\n\t%s is a KEYWORD",yytext);}
"/*" {COMMENT = 1;}
"*/" {COMMENT = 0; cnt++;}
{identifier}\( {if(!COMMENT)printf("\n\nFUNCTION\n\t%s",yytext);}
\{ {if(!COMMENT) printf("\n BLOCK BEGINS");}
\} {if(!COMMENT) printf("\n 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\t%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)
{
if (argc > 1)
{
FILE *file;
file = fopen(argv[1],"r");
if(!file)
{
printf("could not open %s \n",argv[1]);
exit(0);
}
yyin = file;
}
yylex();
printf("\n\n Total No.Of comments are %d",cnt);
return 0;
}
int yywrap()
{
return 1;

}
```

Input:

```
#include<stdio.h>
main()
{
int a,b;
}
```

Output:

```
#include<stdio.h> is a PREPROCESSOR DIRECTIVE
FUNCTION
main (
)
BLOCK BEGINS
int is a KEYWORD
a IDENTIFIER
b IDENTIFIER
BLOCK ENDS
```

3. Source Code to count the numbers of lines, words, spaces, and characters in a given statement

```
/*Lex Program to count numbers of lines, words, spaces and characters

in a given statement*/
%{
#include<stdio.h>
int sc=0,wc=0,lc=0,cc=0;
%}

%%
[\n] { lc++; cc+=yyleng;}
[  \t] { sc++; cc+=yyleng;}
[^\t\n ]+ { wc++;  cc+=yyleng;}
%%

int main(int argc ,char* argv[ ])
{
        printf("Enter the input:\n");
        yylex();
        printf("The number of lines=%d\n",lc);
        printf("The number of spaces=%d\n",sc);
        printf("The number of words=%d\n",wc);
        printf("The number of characters are=%d\n",cc);
}

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

Output:

```
maheshgh@maheshgh:~/LexYacc$ lex demo.l

maheshgh@maheshgh:~/LexYacc$ gcc lex.yy.c

maheshgh@maheshgh:~/LexYacc$ ./a.out

Enter the input: Greetings from VTUPulse.com

Welcome to VTUPulse.com

Lex Yacc Tutorial
```

(Note: After input string press enter and ctrl+d)

```
The number of lines=3

The number of spaces=6

The number of words=9

The number of characters are=70
```

Program to count the numbers of lines, words, spaces, and characters from a given file

```
%{
#include<stdio.h>
int sc=0,wc=0,lc=0,cc=0;
%}

%%
[\n] { lc++; cc+=yyleng;}
[  \t] { sc++; cc+=yyleng;}
[^\t\n ]+ { wc++;  cc+=yyleng;}
%%

int main(int argc ,char* argv[])
{
        if(argc==2)
        {
                yyin=fopen(argv[1],"r");
        }
        else
        {
                printf("Enter the input\n");
                yyin=stdin;
        }
        yylex();
        printf("The number of lines=%d\n",lc);
        printf("The number of spaces=%d\n",sc);
        printf("The number of words=%d\n",wc);
        printf("The number of characters are=%d\n",cc);
}

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

Output:

```
maheshgh@maheshgh:~/LexYacc$ lex demo.l

maheshgh@maheshgh:~/LexYacc$ gcc lex.yy.c

maheshgh@maheshgh:~/LexYacc$ ./a.out input.txt
```

(Note: input.txt is the input file containing input text)

```
The number of lines=3

The number of spaces=6

The number of words=9

The number of characters are=70
```

4. Write a LEX Program to convert the substring abc to ABC from the given input string.

```
%{
#include<stdio.h>
#include<ctype.h>
int i;
%}
%%
[a-z A-Z]* {
          for(i=0;i<=yyleng;i++)
          {
               if((yytext[i]=='a')&&
                         (yytext[i+1]=='b')&&(yytext[i+2]=='c'))
               {
                    yytext[i]='A';
                    yytext[i+1]='B';
                    yytext[i+2]='C';
               }
          }
          printf("%s",yytext);
          }
[\t]*     return;
.* {ECHO;}
\n {printf("%s",yytext);}
%%
main()
{
yylex();
}
int yywrap()
{
return 1;
}
```

OUTPUT:

```
[CSE@localhost ~]$ lex lex1.l

[CSE@localhost ~]$ cc lex.yy.c

[CSE@localhost ~]$. /a.out

 abc

 ABC
```

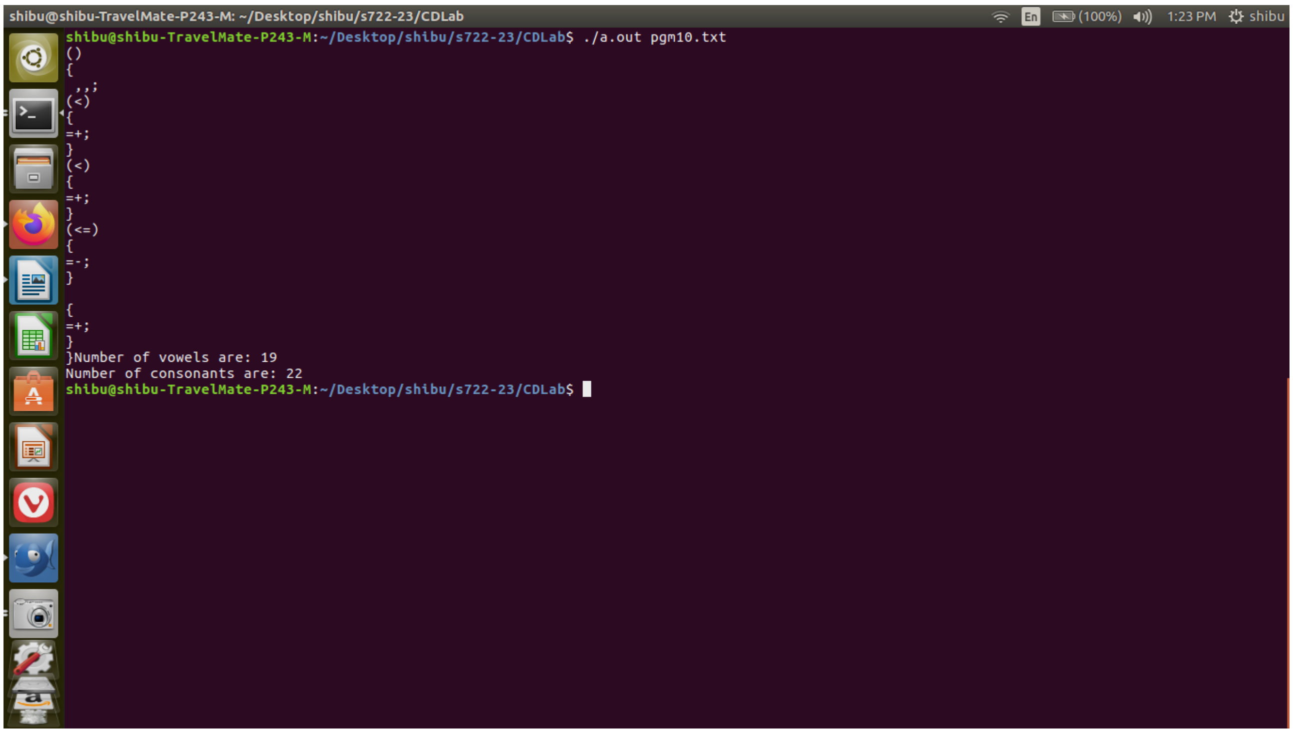
5. LEX program to count the number of vowels and consonants in a given string

Given a string containing both vowels and consonants, write a LEX program to count the number of vowels and consonants in given string.

Approach is very simple. If any vowel is found increase *vowel counter*, if consonant is found increase *consonant counter* otherwise do nothing.

```
%{
	int vow_count=0;
	int const_count =0;
%}

%%
[aeiouAEIOU] {vow_count++;}
[a-zA-Z] {const_count++;}
%%
int yywrap(){}
int main(int argc,char *argv[])
{

	if (argc==2){yyin=fopen(argv[1],”r”);}
	else printf("Enter the string of vowels and consonents:");
	yylex();
	printf("Number of vowels are: %d\n", vow_count);
	printf("Number of consonants are: %d\n", const_count);
	return 0;
}
```

Output



```
shibu@shibu-TravelMate-P243-M: ~/Desktop/shibu/s722-23/CDLab
shibu@shibu-TravelMate-P243-M:~/Desktop/shibu/s722-23/CDLab$ ./a.out pgm10.txt
()
{
 ,,;
(<)
{
=+;
}
(<)
{
=+;
}
(<=)
{
=-;
}

{
=+;
}
}Number of vowels are: 19
Number of consonants are: 22
shibu@shibu-TravelMate-P243-M:~/Desktop/shibu/s722-23/CDLab$ 
```

6. Generate a YACC specification to recognize a valid arithmetic expression that uses operators +, – , *,/ and parenthesis.

AIM:

To write a Yacc program to valid arithmetic expression using Yacc.

ALGORITHM:
Step1: Start the program.
Step2: Reading an expression .
Step3: Checking the validating of the given expression according to the rule using yacc.
Step4: Using expression rule print the result of the given values
Step5: Stop the program.

PROGRAM:

*//Program to recognize a valid arithmetic expression that uses operator +, -, * and /*

LEX PART:

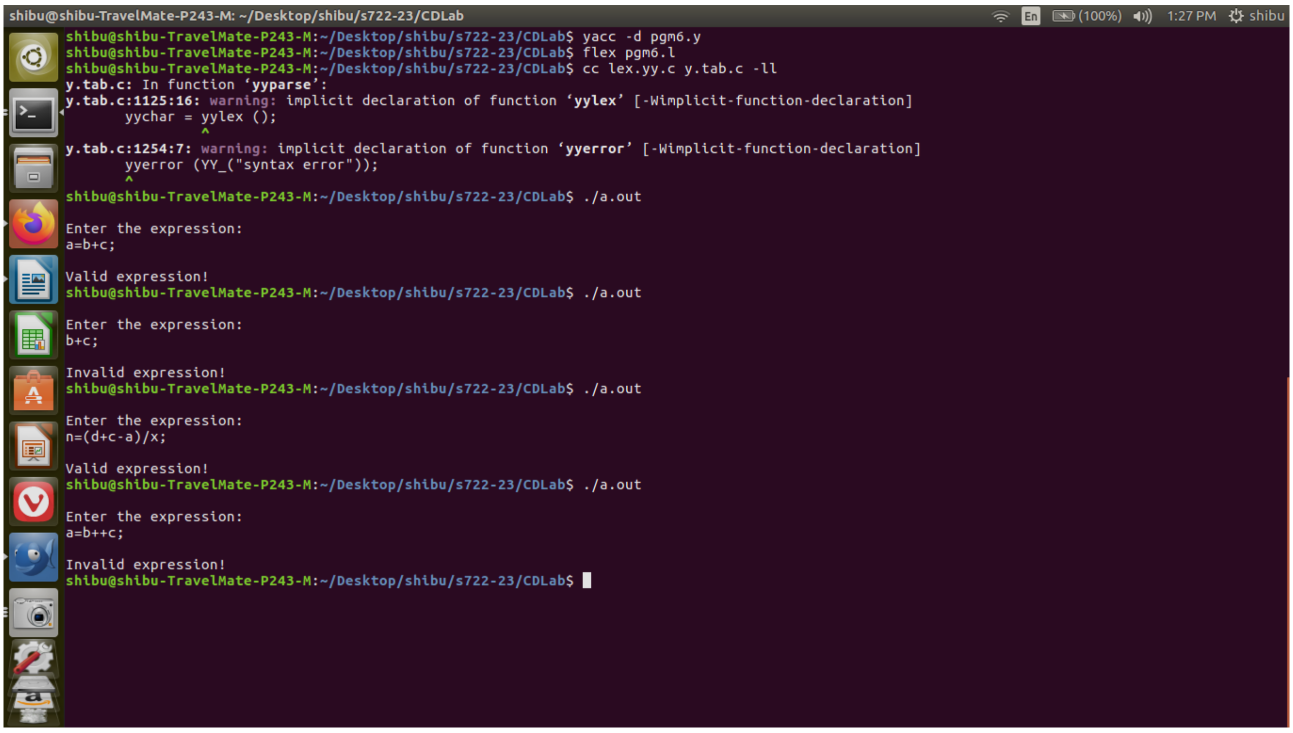
```
%{
 #include "y.tab.h"
%}
%%
[a-zA-Z_][a-zA-Z_0-9]* return id;
[0-9]+(\.[0-9]+)? return num;
[-+/*] return op;
. return yytext[0];
\n return 0;
%%

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:


shibu@shibu-TravelMate-P243-M: ~/Desktop/shibu/s722-23/CDLab
En
(100%)
1:27 PM
shibu

```
shibu@shibu-TravelMate-P243-M:~/Desktop/shibu/s722-23/CDLab$ yacc -d pgm6.y
shibu@shibu-TravelMate-P243-M:~/Desktop/shibu/s722-23/CDLab$ flex pgm6.l
shibu@shibu-TravelMate-P243-M:~/Desktop/shibu/s722-23/CDLab$ cc lex.yy.c y.tab.c -ll
y.tab.c: In function ‘yyparse’:
y.tab.c:1125:16: warning: implicit declaration of function ‘yylex’ [-Wimplicit-function-declaration]
       yychar = yylex ();
                ^
y.tab.c:1254:7: warning: implicit declaration of function ‘yyerror’ [-Wimplicit-function-declaration]
       yyerror (YY_("syntax error"));
       ^
shibu@shibu-TravelMate-P243-M:~/Desktop/shibu/s722-23/CDLab$ ./a.out

Enter the expression:
a=b+c;

Valid expression!
shibu@shibu-TravelMate-P243-M:~/Desktop/shibu/s722-23/CDLab$ ./a.out

Enter the expression:
b+c;

Invalid expression!
shibu@shibu-TravelMate-P243-M:~/Desktop/shibu/s722-23/CDLab$ ./a.out

Enter the expression:
n=(d+c-a)/x;

Valid expression!
shibu@shibu-TravelMate-P243-M:~/Desktop/shibu/s722-23/CDLab$ ./a.out

Enter the expression:
a=b++c;

Invalid expression!
shibu@shibu-TravelMate-P243-M:~/Desktop/shibu/s722-23/CDLab$ 
```

7. Generate a YACC specification to recognize a valid identifier which starts with a letter/ underscore followed by any number of letters or digits.

AIM :

To write a yacc program to check valid variable followed by letter or digits

ALGORITHM:

Step1: Start the program

Step2: Reading an expression

Step3: Checking the validating of the given expression according to the rule using yacc.

Step4: Using expression rule print the result of the given values

Step5: Stop the program

PROGRAM CODE:

//*Program to recognize a valid variable*

LEX PART:

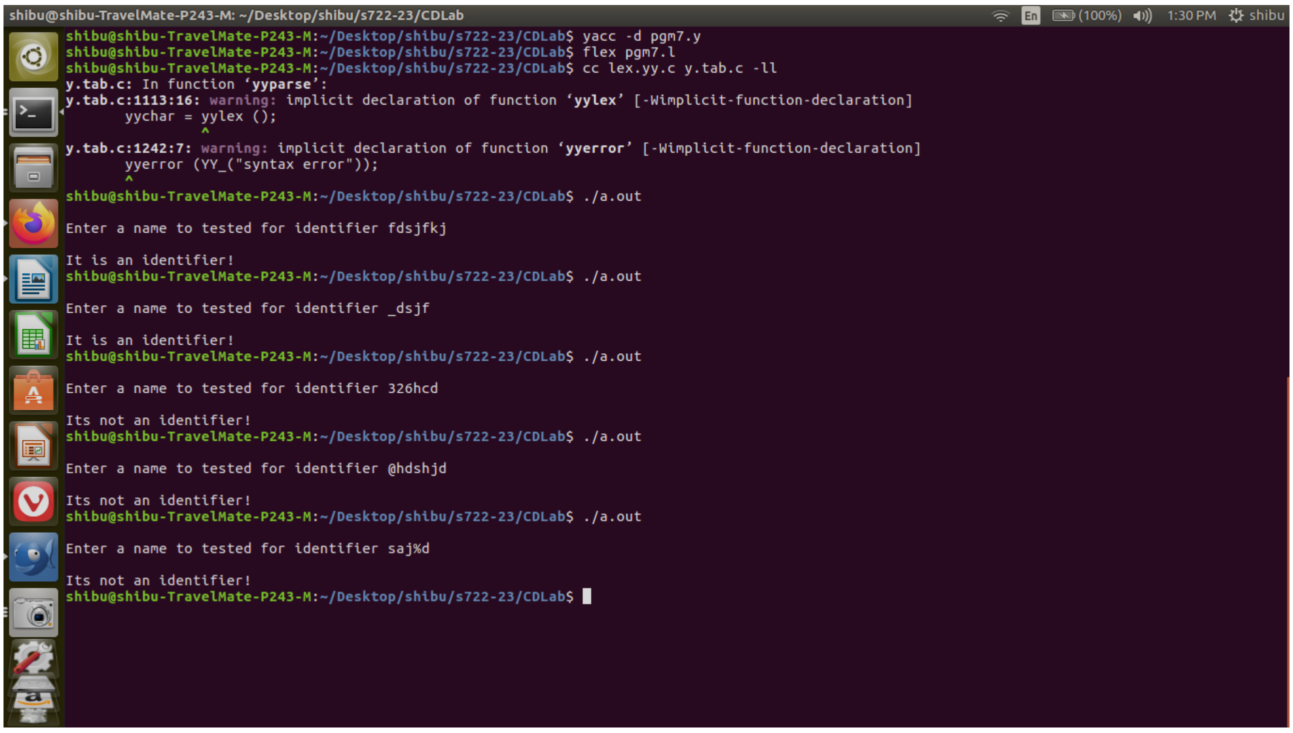
```
%{
 #include "y.tab.h"
%}

%%
[a-zA-Z_] 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 an identifier!\n");
 valid=0;
 return 0;
}

int main()
{
 printf("\nEnter a name to tested for identifier ");
 yyparse();
 if(valid)
 {
 printf("\nIt is an identifier!\n");
 }
}
```

Output:



```
shibu@shibu-TravelMate-P243-M:~/Desktop/shibu/s722-23/CDLab$ yacc -d pgm7.y
shibu@shibu-TravelMate-P243-M:~/Desktop/shibu/s722-23/CDLab$ flex pgm7.l
shibu@shibu-TravelMate-P243-M:~/Desktop/shibu/s722-23/CDLab$ cc lex.yy.c y.tab.c -ll
y.tab.c: In function ‘yyparse’:
y.tab.c:1113:16: warning: implicit declaration of function ‘yylex’ [-Wimplicit-function-declaration]
       yychar = yylex ();
                ^
y.tab.c:1242:7: warning: implicit declaration of function ‘yyerror’ [-Wimplicit-function-declaration]
       yyerror (YY_("syntax error"));
       ^
shibu@shibu-TravelMate-P243-M:~/Desktop/shibu/s722-23/CDLab$ ./a.out

Enter a name to tested for identifier fdsjfkj

It is an identifier!
shibu@shibu-TravelMate-P243-M:~/Desktop/shibu/s722-23/CDLab$ ./a.out

Enter a name to tested for identifier _dsjf

It is an identifier!
shibu@shibu-TravelMate-P243-M:~/Desktop/shibu/s722-23/CDLab$ ./a.out

Enter a name to tested for identifier 326hcd

Its not an identifier!
shibu@shibu-TravelMate-P243-M:~/Desktop/shibu/s722-23/CDLab$ ./a.out

Enter a name to tested for identifier @hdshjd

Its not an identifier!
shibu@shibu-TravelMate-P243-M:~/Desktop/shibu/s722-23/CDLab$ ./a.out

Enter a name to tested for identifier saj%d

Its not an identifier!
shibu@shibu-TravelMate-P243-M:~/Desktop/shibu/s722-23/CDLab$ 
```

8. Implementation of Calculator using LEX and YACC

AIM:

To write a program for implementing a calculator for computing the given expression using semantic rules of the YACC tool and LEX.

ALGORITHM:

Step1: A Yacc source program has three parts as follows:

```
 Declarations
%%
translation rules
 %%
 supporting C routines
```

Step2: Declarations Section: This section contains entries that:

i. Include standard I/O header file.

ii. Define global variables.

iii. Define the list rule as the place to start processing.

iv. Define the tokens used by the parser. v. Define the operators and their precedence.

Step3: Rules Section:
The rules section defines the rules that parse the input stream. Each rule of a grammar production and the associated semantic action.

Step4: Programs Section:
The programs section contains the following subroutines. Because these subroutines are included in this file, it is not necessary to use the yacc library when processing this file.

Step5: main()- The required main program that calls the yyparse subroutine to start the program.

Step6: yyerror(s) -This error-handling subroutine only prints a syntax error message.

Step7: yywrap -The wrap-up subroutine that returns a value of 1 when the end of input occurs. The calc.lex file contains include statements for standard input and output, as programmar file information if we use the -d flag with the yacc command. The y.tab.h file contains definitions for the tokens that the parser program uses.

Step8: calc.lex contains the rules to generate these tokens from the input stream.

PROGRAM CODE:

//*Implementation of calculator using LEX and YACC*

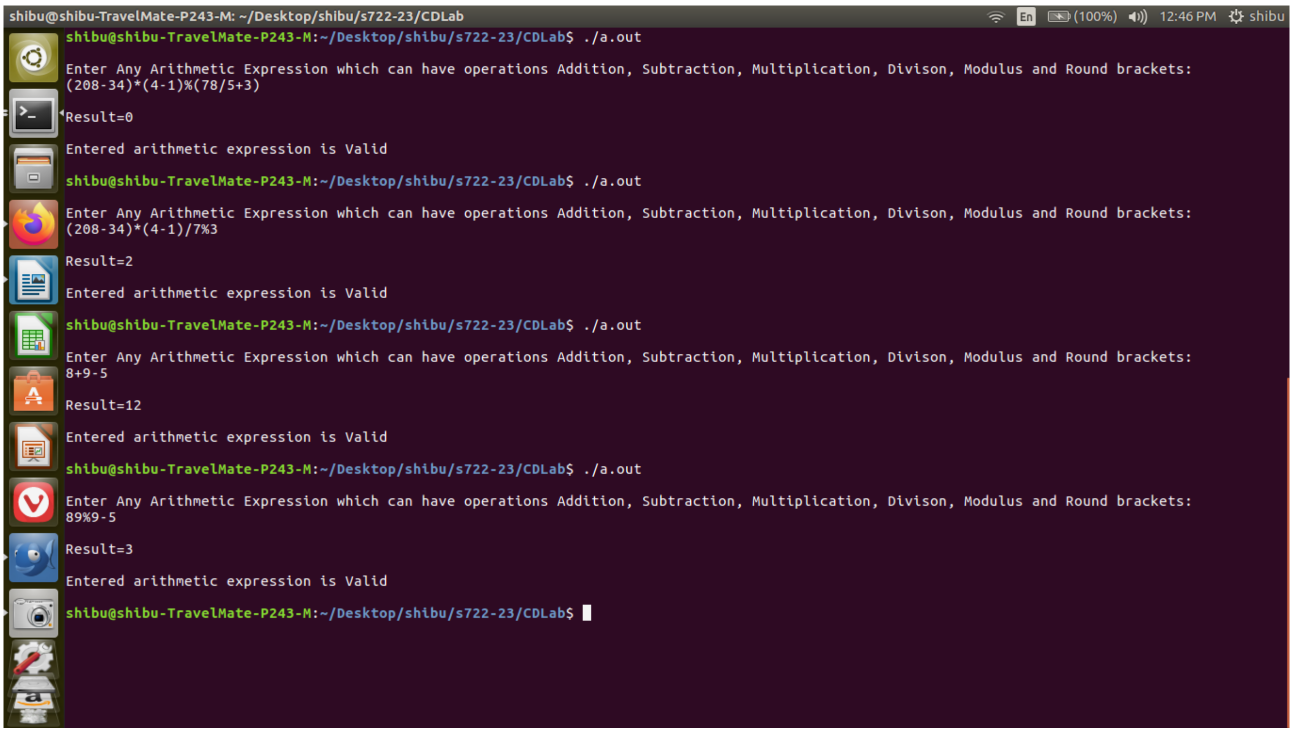
LEX PART:

```
%{
#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


shibu@shibu-TravelMate-P243-M: ~/Desktop/shibu/s722-23/CDLab
En
(100%)
12:46 PM
shibu

```
shibu@shibu-TravelMate-P243-M:~/Desktop/shibu/s722-23/CDLab$ ./a.out

Enter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Divison, Modulus and Round brackets:
(208-34)*(4-1)%(78/5+3)

Result=0

Entered arithmetic expression is Valid

shibu@shibu-TravelMate-P243-M:~/Desktop/shibu/s722-23/CDLab$ ./a.out

Enter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Divison, Modulus and Round brackets:
(208-34)*(4-1)/7%3

Result=2

Entered arithmetic expression is Valid

shibu@shibu-TravelMate-P243-M:~/Desktop/shibu/s722-23/CDLab$ ./a.out

Enter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Divison, Modulus and Round brackets:
8+9-5

Result=12

Entered arithmetic expression is Valid

shibu@shibu-TravelMate-P243-M:~/Desktop/shibu/s722-23/CDLab$ ./a.out

Enter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Divison, Modulus and Round brackets:
89%9-5

Result=3

Entered arithmetic expression is Valid

shibu@shibu-TravelMate-P243-M:~/Desktop/shibu/s722-23/CDLab$
```

9. Convert the BNF rules into YACC form and write code to generate abstract syntax tree.

AIM:

To write a yacc program to change yacc form into abstract syntax Tree

ALGORITHM:
Step1: Reading an expression.
Step2: Calculate the value of given expression
Step3: Display the value of the nodes based on the precedence.
Step4: Using expression rule print the result of the given values

PROGRAM CODE:

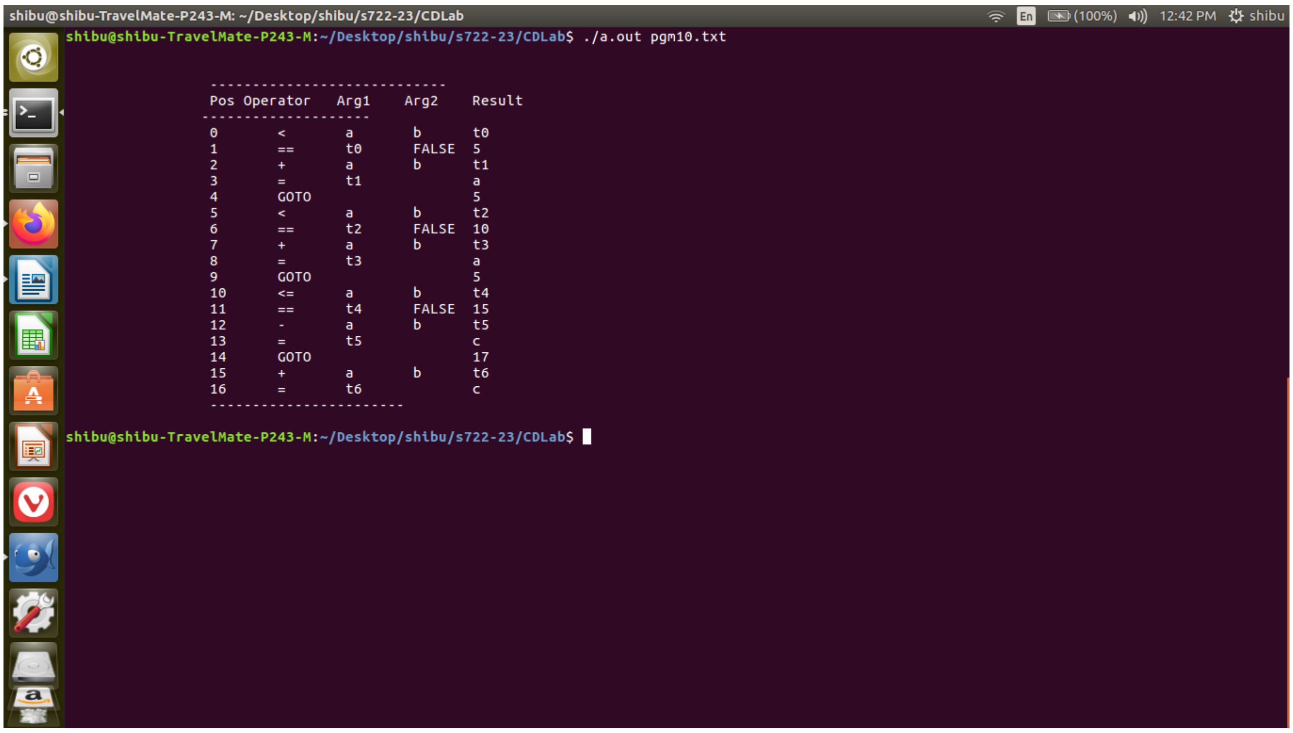
//*Convert the BNF rules into YACC form and*
//*write code to generate Abstract Syntax Tree*

LEX PART:
```
%{
#include"y.tab.h"
#include<stdio.h>
#include<string.h>
int LineNo=1;
%}
identifier [a-zA-Z][_a-zA-Z0-9]*
number [0-9]+|([0-9]*\.[0-9]+)
%%
main\(\) return MAIN;
if return IF;
else return ELSE;
while return WHILE;
int |
char |
float return TYPE;
{identifier} {strcpy(yylval.var,yytext);
return VAR;}
{number} {strcpy(yylval.var,yytext);
return NUM;}
\< |
\> |
\>= |
\<= |
== {strcpy(yylval.var,yytext);
return RELOP;}
[ \t] ;
\n LineNo++;
. return yytext[0];
%%
```
YACC PART:
```
%{
#include<string.h>
#include<stdio.h>
#include<stdlib.h>
struct quad
{
char op[5];
char arg1[10];
char arg2[10];
char result[10];
}QUAD[30];
struct stack
{
int items[100];
int top;
}stk;
int Index=0,tIndex=0,StNo,Ind,tInd;
extern int LineNo;
%}

%union
{
char var[10];
}
%token <var> NUM VAR RELOP
%token MAIN IF ELSE WHILE TYPE
%type <var> EXPR ASSIGNMENT CONDITION IFST ELSEST WHILELOOP
%left '-' '+'
%left '*' '/'
%%
PROGRAM : MAIN BLOCK
;
BLOCK: '{' CODE '}'
;
CODE: BLOCK
| STATEMENT CODE
| STATEMENT
;

STATEMENT: DESCT ';'
| ASSIGNMENT ';'
| CONDST
| WHILEST
;
DESCT: TYPE VARLIST
;
VARLIST: VAR ',' VARLIST
| VAR
;
ASSIGNMENT: VAR '=' EXPR{
strcpy(QUAD[Index].op,"=");
strcpy(QUAD[Index].arg1,$3);
strcpy(QUAD[Index].arg2,"");
strcpy(QUAD[Index].result,$1);
strcpy($$,QUAD[Index++].result);
}
;
EXPR: EXPR '+' EXPR {AddQuadruple("+",$1,$3,$$);}
| EXPR '-' EXPR {AddQuadruple("-",$1,$3,$$);}
| EXPR '*' EXPR {AddQuadruple("*",$1,$3,$$);}
| EXPR '/' EXPR {AddQuadruple("/",$1,$3,$$);}
| '-' EXPR {AddQuadruple("UMIN",$2,"",$$);}
| '(' EXPR ')' {strcpy($$,$2);}
| VAR
| NUM
;
CONDST: IFST{
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);
}
| IFST ELSEST
;
IFST: IF '(' CONDITION ')' {
strcpy(QUAD[Index].op,"==");
strcpy(QUAD[Index].arg1,$3);
strcpy(QUAD[Index].arg2,"FALSE");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
}
BLOCK { strcpy(QUAD[Index].op,"GOTO"); strcpy(QUAD[Index].arg1,"");
strcpy(QUAD[Index].arg2,"");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
};
ELSEST: ELSE{
tInd=pop();
Ind=pop();
push(tInd);
sprintf(QUAD[Ind].result,"%d",Index);
}
BLOCK{
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);
};
CONDITION: VAR RELOP VAR {AddQuadruple($2,$1,$3,$$);
StNo=Index-1;
}
| VAR
| NUM
;
WHILEST: WHILELOOP{
Ind=pop();
sprintf(QUAD[Ind].result,"%d",StNo);
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);
}
;
WHILELOOP: WHILE'('CONDITION ')' {
strcpy(QUAD[Index].op,"==");
strcpy(QUAD[Index].arg1,$3);
strcpy(QUAD[Index].arg2,"FALSE");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
}
BLOCK {
strcpy(QUAD[Index].op,"GOTO");
strcpy(QUAD[Index].arg1,"");
strcpy(QUAD[Index].arg2,"");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
}
;
%%
extern FILE *yyin;
int main(int argc,char *argv[])
{
FILE *fp;
int i;
if(argc>1)
{
fp=fopen(argv[1],"r");
if(!fp)
{
printf("\n File not found");
exit(0);
}
yyin=fp;
}
yyparse();
printf("\n\n\t\t ----------------------------""\n\t\t Pos Operator \tArg1
\tArg2 \tResult" "\n\t\t--------------------");
for(i=0;i<Index;i++)
{
printf("\n\t\t %d\t %s\t %s\t %s\t
%s",i,QUAD[i].op,QUAD[i].arg1,QUAD[i].arg2,QUAD[i].result);
}
printf("\n\t\t -----------------------");
printf("\n\n"); return 0; }
void push(int data)
{ stk.top++;
if(stk.top==100)
{
printf("\n Stack overflow\n");
exit(0);
}
stk.items[stk.top]=data;
}
int pop()
{
int data;
if(stk.top==-1)
{
printf("\n Stack underflow\n");
exit(0);
}
data=stk.items[stk.top--];
return data;
}
void AddQuadruple(char op[5],char arg1[10],char arg2[10],char result[10])
{
strcpy(QUAD[Index].op,op);
strcpy(QUAD[Index].arg1,arg1);
strcpy(QUAD[Index].arg2,arg2);
```

```
sprintf(QUAD[Index].result,"t%d",tIndex++);
strcpy(result,QUAD[Index++].result);
}
yyerror()
{
printf("\n Error on line no:%d",LineNo);
}
```

Input File:

```
main()
{
int a,b,c;
if(a<b)
{
a=a+b;
}
while(a<b)
{
a=a+b;
}
if(a<=b)
{
c=a-b;
}
else
{
c=a+b;
}
}
```



```
shibu@shibu-TravelMate-P243-M: ~/Desktop/shibu/s722-23/CDLab
shibu@shibu-TravelMate-P243-M:~/Desktop/shibu/s722-23/CDLab$ ./a.out pgm10.txt


    ----------------------------
    Pos Operator    Arg1     Arg2     Result
   --------------------
    0         <          a        b        t0
    1         ==         t0       FALSE    5
    2         +          a        b        t1
    3         =          t1                a
    4         GOTO                         5
    5         <          a        b        t2
    6         ==         t2       FALSE    10
    7         +          a        b        t3
    8         =          t3                a
    9         GOTO                         5
    10        <=         a        b        t4
    11        ==         t4       FALSE    15
    12        -          a        b        t5
    13        =          t5                c
    14        GOTO                         17
    15        +          a        b        t6
    16        =          t6                c
    -----------------------

shibu@shibu-TravelMate-P243-M:~/Desktop/shibu/s722-23/CDLab$
```

10. Write a program to find ε – closure of all states of any given NFA with ε transition.

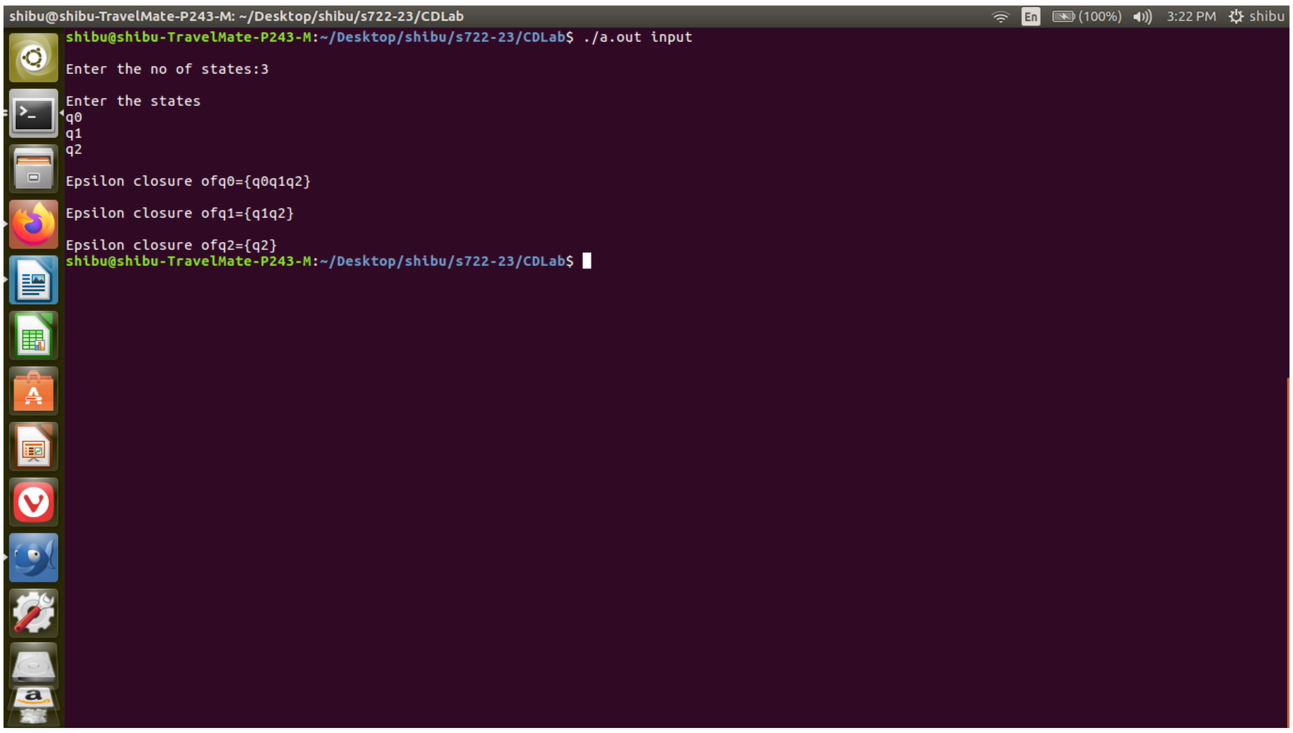
```
#include<stdio.h>
#include<string.h>

char result[20][20],copy[3],states[20][20];

void add_state(char a[3],int i){
strcpy(result[i],a);
}

void display(int n){
int k=0;
printf("\nEpsilon closure of %s={",copy);
while(k<n){
     printf("%s",result[k]);
     k++;
     }
printf("}\n");
}
int main(int argc,char *argv[]){
FILE* INPUT;
INPUT=fopen(argv[1],"r");
char state[3];
int end,i=0,n,k=0;
char state1[3],input[3],state2[3];
printf("\nEnter the no of states:");
scanf("%d",&n);
printf("\nEnter the states\n");
for(k=0;k<3;k++){
     scanf("%s",states[k]);
     }

for(k=0;k<n;k++){
     i=0;
     strcpy(state,states[k]);
     strcpy(copy,state);
     add_state(state,i++);
     while(1){
          end=fscanf(INPUT,"%s%s%s",state1,input,state2);
          if(end==EOF){
               break;
               }
          if(strcmp(state,state1)==0){
               if(strcmp(input,"e")==0){
                    add_state(state2,i++);
                    strcpy(state,state2);
                    }
               }
          }
     display(i);
     rewind(INPUT);
     }
return 0;
}
```

Input-

```
q0 0 q0
q0 1 q1
q0 e q1
q1 1 q2
q1 e q2
```

Output-

```
Enter the no of states: 3
Enter the states
q0
q1
q2
Epsilon closure of q0 = { q0 q1 q2 }
Epsilon closure of q1 = { q1 q2 }
Epsilon closure of q2 = { q2 }
```

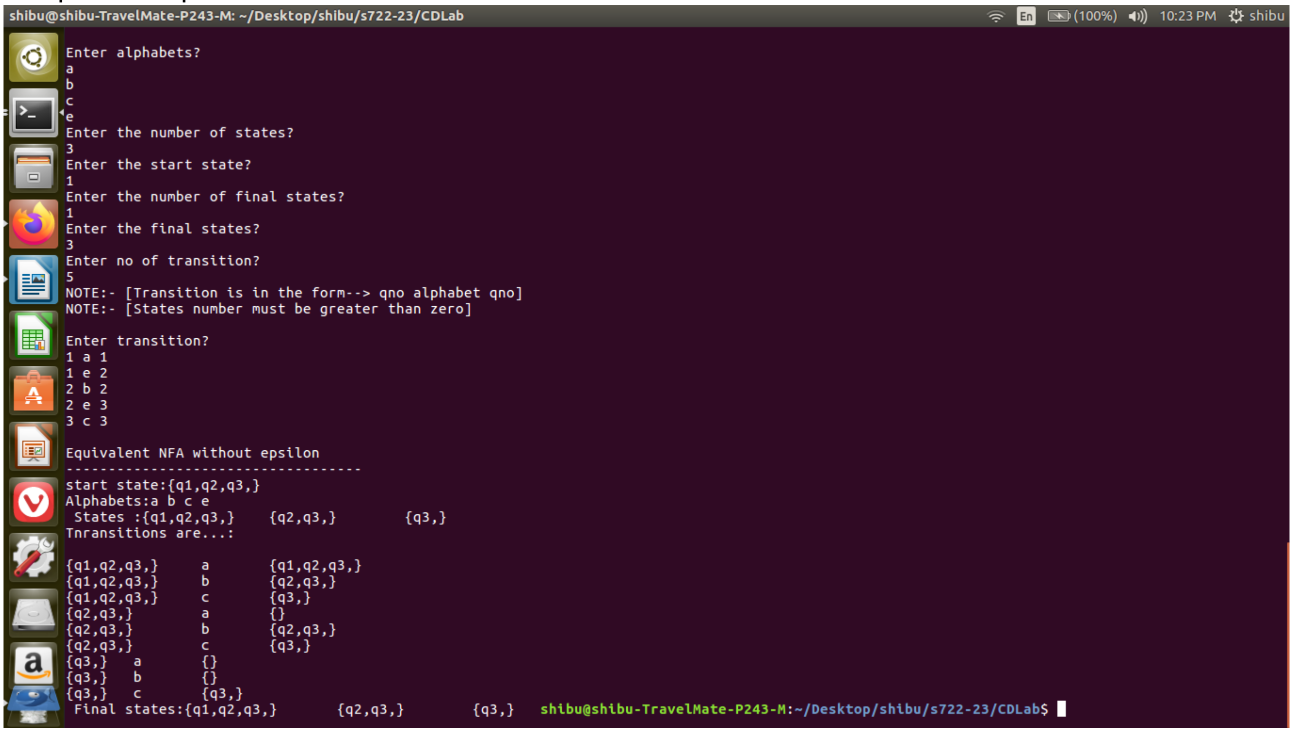


```
shibu@shibu-TravelMate-P243-M: ~/Desktop/shibu/s722-23/CDLab
shibu@shibu-TravelMate-P243-M:~/Desktop/shibu/s722-23/CDLab$ ./a.out input

Enter the no of states:3

Enter the states
q0
q1
q2

Epsilon closure ofq0={q0q1q2}

Epsilon closure ofq1={q1q2}

Epsilon closure ofq2={q2}
shibu@shibu-TravelMate-P243-M:~/Desktop/shibu/s722-23/CDLab$
```

11. Write a program to convert NFA with ε transition to NFA without ε transition.

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int st;
    struct node *link;
};

void findclosure(int,int);
void insert_trantbl(int ,char, int);
int findalpha(char);
void findfinalstate(void);
void unionclosure(int);
void print_e_closure(int);
static int
set[20],nostate,noalpha,s,notransition,nofinal,start,finalstate[20],c,r,b
uffer[20];
char alphabet[20];
static int e_closure[20][20]={0};
struct node * transition[20][20]={NULL};
void main()
{
     int i,j,k,m,t,n;

     struct node *temp;
     printf("enter the number of alphabets?\n");
     scanf("%d",&noalpha);
     getchar();
     printf("NOTE:- [ use letter e as epsilon]\n");
     printf("NOTE:- [e must be last character ,if it is present]\n");
     printf("\nEnter alphabets?\n");
     for(i=0;i<noalpha;i++){
           alphabet[i]=getchar();
           getchar();
           }
      printf("Enter the number of states?\n");
      scanf("%d",&nostate);
      printf("Enter the start state?\n");
      scanf("%d",&start);
      printf("Enter the number of final states?\n");
      scanf("%d",&nofinal);
      printf("Enter the final states?\n");
      for(i=0;i<nofinal;i++)
           scanf("%d",&finalstate[i]);
      printf("Enter no of transition?\n");
      scanf("%d",&notransition);
      printf("NOTE:-[Transition is in the form-->qno alphabet
           no]\n",notransition);
      printf("NOTE:- [States number must be greater than zero]\n");
      printf("\nEnter transition?\n");
      for(i=0;i<notransition;i++){
           scanf("%d %c%d",&r,&c,&s);
           insert_trantbl(r,c,s);
          }
    printf("\n");
    for(i=1;i<=nostate;i++){
          c=0;
          for(j=0;j<20;j++){
              buffer[j]=0;
              e_closure[i][j]=0;
               }
        findclosure(i,i);
          }
    printf("Equivalent NFA without epsilon\n");
    printf("-------------------------------------\n");
    printf("start state:");
    print_e_closure(start);
    printf("\nAlphabets:");
    for(i=0;i<noalpha;i++)
    printf("%c ",alphabet[i]);
    printf("\n States :" );
    for(i=1;i<=nostate;i++)
    print_e_closure(i);
    printf("\nTnransitions are...:\n");
    for(i=1;i<=nostate;i++)
    {
         for(j=0;j<noalpha-1;j++)
        {
             for(m=1;m<=nostate;m++)
                    set[m]=0;
             for(k=0;e_closure[i][k]!=0;k++)
             {
                  t=e_closure[i][k];
                 temp=transition[t][j];
                 while(temp!=NULL)
                 {
                      unionclosure(temp->st);
                      temp=temp->link;
                 }
            }
            printf("\n");
            print_e_closure(i);
            printf("%c\t",alphabet[j] );
            printf("{");
            for(n=1;n<=nostate;n++)
            {
                  if(set[n]!=0)
                      printf("q%d,",n);
            }
            printf("}");
        }
    }
    printf("\n Final states:");
    findfinalstate();
}

void findclosure(int x,int sta)
{
      struct node *temp;
      int i;
     if(buffer[x])
          return;
      e_closure[sta][c++]=x;
      buffer[x]=1;
      if(alphabet[noalpha-1]=='e' && transition[x][noalpha-1]!=NULL)
        {
              temp=transition[x][noalpha-1];
              while(temp!=NULL)
              {
                    findclosure(temp->st,sta);
                    temp=temp->link;
              }
        }
 }

void insert_trantbl(int r,char c,int s)
{
     int j;
     struct node *temp;
      j=findalpha(c);
     if(j==999)
     {
          printf("error\n");
          exit(0);
     }
    temp=(struct node *) malloc(sizeof(struct node));
    temp->st=s;
    temp->link=transition[r][j];
    transition[r][j]=temp;
}

int findalpha(char c)
{
      int i;
      for(i=0;i<noalpha;i++)
         if(alphabet[i]==c)
             return i;
        return(999);
}

void unionclosure(int i)
{
       int j=0,k;
      while(e_closure[i][j]!=0)
      {
           k=e_closure[i][j];
           set[k]=1;
           j++;
      }
}
void findfinalstate()
{
      int i,j,k,t;
      for(i=0;i<nofinal;i++)
     {
           for(j=1;j<=nostate;j++)
           {
               for(k=0;e_closure[j][k]!=0;k++)
                {
                    if(e_closure[j][k]==finalstate[i])
                    {
                        print_e_closure(j);
                    }
               }
           }
      }


 }

void print_e_closure(int i)
{
    int j;
    printf("{");
    for(j=0;e_closure[i][j]!=0;j++)
            printf("q%d,",e_closure[i][j]);
    printf("}\t");
}
```

sample output



```
shibu@shibu-TravelMate-P243-M: ~/Desktop/shibu/s722-23/CDLab

Enter alphabets?
a
b
c
e
Enter the number of states?
3
Enter the start state?
1
Enter the number of final states?
1
Enter the final states?
3
Enter no of transition?
5
NOTE:- [Transition is in the form--> qno alphabet qno]
NOTE:- [States number must be greater than zero]

Enter transition?
1 a 1
1 e 2
2 b 2
2 e 3
3 c 3

Equivalent NFA without epsilon
-----------------------------------
start state:{q1,q2,q3,}
Alphabets:a b c e
 States :{q1,q2,q3,}    {q2,q3,}        {q3,}
Tnransitions are...:

{q1,q2,q3,}     a       {q1,q2,q3,}
{q1,q2,q3,}     b       {q2,q3,}
{q1,q2,q3,}     c       {q3,}
{q2,q3,}        a       {}
{q2,q3,}        b       {q2,q3,}
{q2,q3,}        c       {q3,}
{q3,}   a       {}
{q3,}   b       {}
{q3,}   c       {q3,}
 Final states:{q1,q2,q3,}       {q2,q3,}        {q3,}   shibu@shibu-TravelMate-P243-M:~/Desktop/shibu/s722-23/CDLab$
```

12. Write a program to convert NFA to DFA

```
#include<stdio.h>

int Fa[10][10][10],states[2][10],row=0,col=0,sr=0,sc=0,th=0,
in,stat,new_state[10][10],max_inp=-1,no_stat;
FILE *fp;

int search(int search_var)
 {

   int i;
   for(i=0;i<no_stat;i++)
      if(search_var == states[1][i])
      return 1;
   return 0;
 }


int sort(int *arr,int count)
 {
   int temp,i,j;
   for(i=0;i<count-1;i++)
     {
      for(j=i+1;j<count;j++)
    {
      if(arr[i]>=arr[j])
        {
          temp=arr[i];
          arr[i]=arr[j];
          arr[j]=temp;
         }
     }
     }
   return 0;
 }

int checkcon(int *arr,int *count)  //for doing this {4,1}={1,2,1}=={1,2}
 {
   int i,temp,j,k,c,t,m;
   for(i=0;i<*count;i++)
     {
       if(arr[i]>row)
      {
         temp =arr[i];
         c=0;
         t=0;
        while(new_state[arr[i]][t]!=-1)
         {
           t++;
           c++;
         }
        //right shift from ith postion (c-2) th time
        for(k=0;k<=c-2;k++)
           {
        for(j=9;j>=i+1+k;j--)
         {
          arr[j]=arr[j-1];
         }
           }

         t=0;
         for(j=i;j<c;j++)
        {
           arr[j]=new_state[temp][t];
           t++;
         }
       }
     }
   c=0;
  for(i=0;arr[i]!=-1;i++)
      c++;
   *count=c;
   return 0;
 }

int remove_duplicate(int *arr,int *count)
 {
   int i,j=0;
   for(i=1;i<*count;i++)
     {
      if(arr[i]!=arr[j])
     {
       j++;
       arr[j]=arr[i];
     }
     }
   *count=j+1;
   return 0;
 }

int check(int i ,int j,int c,int *name)///for checking is this a new
state?
  {
    int t,l,f;
    for(l=0;l<=stat;l++)
      {
       t=0; f=0;
       while(Fa[i][j][t]!=-1)
     {
        if(Fa[i][j][t]==new_state[l][t])
          t++;
        else
          {
        f=1;
        break;
         }
     }
      if((t==c)&&!f)
    {
     *name=l;
     return 1;
    }
    }
    return 0;
}
```

```
int trans(int i ,int j,int t,int c,int *count,int *arr)//transition o/p
for particular i/p on states
 {
   int k=0,co,temp;
   *count=0;
   for(k=0;k<c;k++)
     {
       temp=Fa[i][j][k];
       co=0;
       while(Fa[temp][t][co]!=-1)
      {
        arr[*count]=Fa[temp][t][co++];
        (*count)++;
      }
     }
return 0;
 }

int nfa2dfa(int start,int end)
 {
   int j,t,c,i,k,count,arr[10],name,l;
   for(i=start;i<=end;i++)
      {
    for(j=0;j<=max_inp;j++)
      {
        c=0;t=0;
        while(Fa[i][j][t]>=0)
           {
         t++;
         c++;
           }
          if(c>1)
        {
         if(check(i,j,c,&name)==0)
           {
             for(k=0;k<c;k++)
              {
             new_state[stat][k]=Fa[i][j][k];
             for(l=0;states[1][l]!=-1;l++)
                if(new_state[stat][k] == states[1][l]&& !search(stat))
                  states[1][no_stat++]=stat;
              }

              for(t=0;t<=max_inp;t++)
            {
               count=0;
               for(k=0;k<10;k++)
                  arr[k]=-1;
               trans(i,j,t,c,&count,arr);


               checkcon(arr,&count);

               sort(arr,count);
               remove_duplicate(arr,&count);

               for(k=0;k<count;k++)
                 Fa[stat][t][k]=arr[k];
            }
          Fa[i][j][0]=stat++;
          for(t=1;t<c;t++)
            Fa[i][j][t]=-1;
         }
           else
         {
           Fa[i][j][0]=name ;
           for(t=1;t<c;t++)
             Fa[i][j][t]=-1;
         }
          }
       }

     }
  return 0;
 }

int main()
 {

     int i,j,k,flag=0,start,end;
     char c,ch;
     fp=fopen("Nfa_ip.txt","r+");


     for(i=0;i<2;i++)
       for(j=0;j<10;j++)
     states[i][j]=-1;

    for(i=0;i<10;i++)
       for(j=0;j<10;j++)
     new_state[i][j]=-1;

     for(i=0;i<10;i++)
       for(j=0;j<10;j++)
      for(k=0;k<10;k++)
          Fa[i][j][k]=-1;

     while(fscanf(fp,"%d",&in)!=EOF)
       {
       fscanf(fp,"%c",&c);

        if(flag)
          {
         states[sr][sc++]=in;
         if(c=='\n')
            {
              sr++;
              sc=0;
            }
          }
        else if(c=='#')
          {
         flag=1;
         Fa[row][col][th]=in;

          }
         else if(!flag)
```

```
          {
         Fa[row][col][th]=in;
         if(c==',')
           {
             th++;
           }
         else if(c=='\n')
           {
             if(max_inp<col)
               max_inp=col;
              col=0;
              row++;
              th=0;
          }
        else if(c!=',')
         {
             col++;
             th=0;
          }
          }

      }
   no_stat=0;
   i=0;
   while(states[1][i++]!=-1)
      no_stat++;
      stat=row+1;
      start=0;end=row;
      while(1)
      {
         nfa2dfa(start,end);
         start=end+1;
         end=row;
         if(start>end)
          break;
      }

     printf("\n\nDFA IS : \n\n\n");
     for(i=0;i<=max_inp;i++)
     printf("\t%d",i);
     printf("\n");
     printf("----------------------------\n");

     for(i=0;i<stat;i++)
       {
     printf("%d-> |",i);
       for(j=0;j<=max_inp;j++)
      {
        printf("%2d        ",Fa[i][j][0]);
      }
      printf("\n");
       }
printf("\n\n");
printf("Total Number Of State Is : %d \n\n",stat);
printf("Final States Are : ");
  for(i=0;states[1][i]!=-1;i++)
     printf("%d ",states[1][i]);
printf("\n\n");
getch();
    return 0;
}
```

13. Write a program to minimize any given DFA.

C program for construction of minimized DFA from a given regular expression.

OBJECTIVE: write a program for construction of minimized DFA from a given regular expression using C.

ALGORITHM:

1. Get the start state, final state, input symbols as input and also give the edge value for each state.
2. Maintain a stack required for transition from one state to other state.
3. Using Pop or push function perform the insertion and deletion of elements when required.
4. Finally conversion has been made to change from regular expression to minimized DFA and the output is displayed as DFA transition table.

PROGRAM:

```
#include

#include

#define STATES 50

struct Dstate

{

char name;

char StateString[STATES+1];

char trans[10];

int is_final;

}Dstates[50];

struct tran

{

char sym;

int tostates[50];

int notran;

};

struct state

{

int no;
```

```
struct tran tranlist[50];
};
int stackA[100],stackB[100],c[100],Cptr=-1,Aptr=-1,Bptr=-1;
struct state States[10];
char temp[STATES+1],inp[10];
int nos,noi,nof,j,k,nods=-1;

void pushA(int z)
{
stackA[++Aptr]=z;
}
void pushB(int z)
{
stackB[++Bptr]=z;
}

int popA()
{
return stackA[Aptr--];
}
void copy(int i)
{
char temp[STATES+1]=" ";
int k=0;
Bptr=-1;
strcpy(temp,Dstates[i].StateString);
while(temp[k]!='\0')
{
pushB(temp[k]-'0');
k++;
```

```
}
}
int popB()
{
return stackB[Bptr--];
}
int peekA()
{
return stackA[Aptr];
}
int peekB()
{
return stackA[Bptr];
}
int seek(int arr[],int ptr,int s)
{
int i;
for(i=0;i<=ptr;i++)
{
if(s==arr[i])
return 1;
}
return 0;
}
void sort()
{
int i,j,temp;
for(i=0;i
{
```

```
for(j=0;j<(Bptr-i);j++)
{
if(stackB[j]>stackB[j+1])
{
temp=stackB[j];
stackB[j]=stackB[j+1];
stackB[j+1]=temp;
}
}
 }
 }
void tostring()
{
int i=0;
sort();
for(i=0;i<=Bptr;i++)
{
temp[i]=stackB[i]+'0';
}
temp[i]='\0';
}
void display_DTran()
{
int i,j;
printf("\n\t\t DFA transition table");
printf("\n\t\t ---------------------------------------------- ");
printf("\n States \tString \tInputs\n");
for(i=0;i
{
```

```
printf("\t %c",inp[i]);
}
printf("\n\t ------------------------------------------------ ");
for(i=0;i
{
if(Dstates[i].is_final==0)
printf("\n%c",Dstates[i].name);
else
printf("\n*%c",Dstates[i].name);
printf("\t%s",Dstates[i].StateString);
for(j=0;j
{
printf("\t%c",Dstates[i].trans[j]);
}
 }
 printf("\n");
}
void move(int st,int j)
{
int ctr=0;
while(ctr
{
pushA(States[st].tranlist[j].tostates[ctr++]);
}
}
void lambda_closure(int st)
{
int ctr=0,in_state=st,curst=st,chk;
while(Aptr!=-1)
```

```
{
curst=popA();
ctr=0;
in_state=curst;
while(ctr<=States[curst].tranlist[noi].notran)
{
chk=seek(stackB,Bptr,in_state);
if(chk==0)
pushB(in_state);
in_state=States[curst].tranlist[noi].tostates[ctr++];
chk=seek(stackA,Aptr,in_state);
if(chk==0 && ctr<=States[curst].tranlist[noi].notran)
pushA(in_state);
}
 }
}
void main()
{
int i,final[20],start,fin=0;
char c,ans,st[20];
printf("\n Enter no of states in NFA:");
scanf("%d",&nos);
for(i=0;i
{
States[i].no=i;
}
printf("\n Enter the start states:");
scanf("%d",&start);
printf("Enter the no of final states:");
```

```
scanf("%d",&nof);

printf("Enter the final states:\n");

for(i=0;i

scanf("%d",&final[i]);

printf("\n Enter the no of input symbols:");

scanf("%d",&noi);

c=getchar();

printf("Enter the input symbols:\n");

for(i=0;i

{

scanf("%c",&inp[i]);

c=getchar();

}

//g1inp[i]='e';

inp=[i]=’e’;

printf("\n Enter the transitions:(-1 to stop)\n");

for(i=0;i

{

for(j=0;j<=noi;j++)

{

States[i].tranlist[j].sym=inp[j];

k=0;

ans='y';

while(ans=='y')

{

printf("move(%d,%c);",i,inp[j]);

scanf("%d",&States[i].tranlist[j].tostates[k++]);

if((States[i].tranlist[j].tostates[k-1]==-1))

{
```

```
k--;
ans='n';
break;
 }
 }
States[i].tranlist[j].notran=k;
 }
}
i=0;nods=0,fin=0;
pushA(start);
lambda_closure(peekA());
tostring();
Dstates[nods].name='A';
nods++;
strcpy(Dstates[0].StateString,temp);
while(i
{
for(j=0;j
{
fin=0;
copy(i);
while(Bptr!=-1)
{
move(popB(),j);
}


while(Aptr!=-1)
lambda_closure(peekA());
```

```
tostring();
for(k=0;k
{
if((strcmp(temp,Dstates[k].StateString)==0))
{
Dstates[i].trans[j]=Dstates[k].name;
break;
}
}
if(k==nods)
{
nods++;
for(k=0;k
{
fin=seek(stackB,Bptr,final[k]);
if(fin==1)
{
Dstates[nods-1].is_final=1;
break;
}
 }
strcpy(Dstates[nods-1].StateString,temp);
Dstates[nods-1].name='A'+nods-1;
Dstates[i].trans[j]=Dstates[nods-1].name;
}
}
i++;
}
display_DTran();
```

```
}
```

OUTPUT:

```
Enter the no of input symbols:2

Enter the input symbols:

a ,b

Enter the transitions:(-1 to stop)

move(0,a);-1

move(0,b);-1

move(0,e);1

move(0,e);7

move(0,e);-1

move(1,a);-1

move(1,b);-1

move( 1,e);2

move(1,e);4

move(1,e);-1

move(2,a);3

move(2,a);3

move(2,a);-1

move(2,b);-1

move(2,e);-1

move(3,a);-1

move(3,b);-1

move(3,e);6

move(3,e);-1

move(4,a);-1

move(4,b);-1

move(4,e);-1

move(5,a);-1
move(5,b);-1

move(5,e);6

move(5,e);1

move(5,e);-1

move(6,a);-1

move(6,b);-1

move(6,e);-1

move(7,a);-1

move(7,b);-1

move(7,e);-1
```

DFA transition table

```
States String Inputs

 a b

 -------------------------------------------------

A  01247 B C

B  36 C C

C  C C
```

14. Write a program to find First and Follow of any given grammar

```
#include<stdio.h>
```

```
#include<string.h>

int i,j,l,m,n=0,o,p,nv,z=0,x=0;
char str[10],temp,temp2[10],temp3[20],*ptr;

struct prod
{
    char lhs[10],rhs[10][10],ft[10],fol[10];
    int n;
}pro[10];

void findter()
{
    int k,t;
    for(k=0;k<n;k++)
    {
        if(temp==pro[k].lhs[0])
        {
            for(t=0;t<pro[k].n;t++)
            {
                if( pro[k].rhs[t][0]<65 || pro[k].rhs[t][0]>90 )
                    pro[i].ft[strlen(pro[i].ft)]=pro[k].rhs[t][0];
                else if( pro[k].rhs[t][0]>=65 && pro[k].rhs[t]
[0]<=90 )
                {
                    temp=pro[k].rhs[t][0];
                    if(temp=='S')
                        pro[i].ft[strlen(pro[i].ft)]='#';
                    findter();
                }
            }
            break;
        }
    }
}

void findfol()
{
    int k,t,p1,o1,chk;
    char *ptr1;
    for(k=0;k<n;k++)
    {
        chk=0;
        for(t=0;t<pro[k].n;t++)
        {
            ptr1=strchr(pro[k].rhs[t],temp);
            if( ptr1 )
            {
                p1=ptr1-pro[k].rhs[t];
                if(pro[k].rhs[t][p1+1]>=65 && pro[k].rhs[t]
[p1+1]<=90)
                {
                    for(o1=0;o1<n;o1++)
                        if(pro[o1].lhs[0]==pro[k].rhs[t][p1+1])
                        {
                                strcat(pro[i].fol,pro[o1].ft);
                                chk++;
                        }
                }
                else if(pro[k].rhs[t][p1+1]=='\0')
                {
                    temp=pro[k].lhs[0];
                    if(pro[l].rhs[j][p]==temp)
                        continue;
                    if(temp=='S')
                        strcat(pro[i].fol,"$");
                    findfol();
                    chk++;
                }
                else
                {
                    pro[i].fol[strlen(pro[i].fol)]=pro[k].rhs[t]
[p1+1];
                    chk++;
                }
            }
        }
        if(chk>0)
            break;
    }
}

int main()
{
    FILE *f;
    //clrscr();
    for(i=0;i<10;i++)
        pro[i].n=0;

    f=fopen("tab5.txt","r");
    while(!feof(f))
    {
        fscanf(f,"%s",pro[n].lhs);
        if(n>0)
        {
            if( strcmp(pro[n].lhs,pro[n-1].lhs) == 0 )
            {
                pro[n].lhs[0]='\0';
                fscanf(f,"%s",pro[n-1].rhs[pro[n-1].n]);
                pro[n-1].n++;
                continue;
            }
        }
        fscanf(f,"%s",pro[n].rhs[pro[n].n]);
        pro[n].n++;
        n++;
    }

    printf("\n\nTHE GRAMMAR IS AS FOLLOWS\n\n");
    for(i=0;i<n;i++)
        for(j=0;j<pro[i].n;j++)
            printf("%s -> %s\n",pro[i].lhs,pro[i].rhs[j]);

    pro[0].ft[0]='#';
    for(i=0;i<n;i++)
    {
        for(j=0;j<pro[i].n;j++)
        {
            if( pro[i].rhs[j][0]<65 || pro[i].rhs[j][0]>90 )
            {
                pro[i].ft[strlen(pro[i].ft)]=pro[i].rhs[j][0];
            }
            else if( pro[i].rhs[j][0]>=65 && pro[i].rhs[j]
[0]<=90 )
            {
                temp=pro[i].rhs[j][0];
                if(temp=='S')
                    pro[i].ft[strlen(pro[i].ft)]='#';
                findter();
            }
        }
    }

    printf("\n\nFIRST\n");
    for(i=0;i<n;i++)
    {
        printf("\n%s -> ",pro[i].lhs);
        for(j=0;j<strlen(pro[i].ft);j++)
        {
            for(l=j-1;l>=0;l--)
                if(pro[i].ft[l]==pro[i].ft[j])
                    break;
            if(l==-1)
                printf("%c",pro[i].ft[j]);
        }
    }

    for(i=0;i<n;i++)
        temp2[i]=pro[i].lhs[0];
    pro[0].fol[0]='$';
    for(i=0;i<n;i++)
    {
        for(l=0;l<n;l++)
        {
            for(j=0;j<pro[i].n;j++)
            {
                ptr=strchr(pro[l].rhs[j],temp2[i]);
                if( ptr )
                {
                    p=ptr-pro[l].rhs[j];
                    if(pro[l].rhs[j][p+1]>=65 && pro[l].rhs[j]
[p+1]<=90)
                    {
                        for(o=0;o<n;o++)
                            if(pro[o].lhs[0]==pro[l].rhs[j][p+1])
                                    strcat(pro[i].fol,pro[o].ft);
                    }
                    else if(pro[l].rhs[j][p+1]=='\0')
                    {
                        temp=pro[l].lhs[0];
                        if(pro[l].rhs[j][p]==temp)
```

```
                                        continue;
                                    if(temp=='S')
                                        strcat(pro[i].fol,"$");
                                    findfol();
                                }
                                else

pro[i].fol[strlen(pro[i].fol)]=pro[l].rhs[j][p+1];
                            }
                        }
                    }
                }

    printf("\n\nFOLLOW\n");
    for(i=0;i<n;i++)
    {
        printf("\n%s -> ",pro[i].lhs);
        for(j=0;j<strlen(pro[i].fol);j++)
        {
            for(l=j-1;l>=0;l--)
                if(pro[i].fol[l]==pro[i].fol[j])
                    break;
            if(l==-1)
                printf("%c",pro[i].fol[j]);
        }
    }
    printf("\n");
    //getch();
}
```

Input File(tab5.txt) For First and Follow Program:-

```
S ABE
S a
A p
A t
B Aq
S f
A w
```

15. Design and implement a recursive descent parser for a given grammar.

C program to Construct of recursive descent parsing for the following grammar

E->TE’

E’->+TE/@

T->FT’

T`->*FT’/@

F->(E)/id where @ represents null character

LOGIC: Read the input string. Write procedures for the non terminals Verify the next token equals to non terminals if it satisfies match the non terminal. If the input string does not match print error.

PROGRAM:

```
#include

#include

#include

char input[100];

int i,l;

void main()

{

clrscr();

printf("\nRecursive descent parsing for the following grammar\n");
printf("\nE->TE'\nE'->+TE'/@\nT->FT'\nT'->*FT'/@\nF->(E)/ID\n");
printf("\nEnter the string to be checked:"); gets(input);

if(E())

{

if(input[i+1]=='\0')
```

```
printf("\nString is accepted");

else

printf("\nString is not accepted");

}

else

printf("\nString not accepted");

getch();

}

E()

{

if(T())

{

if(EP())

return(1);

else

return(0);

}

else

return(0);

}

EP()

{

if(input[i]=='+')

{

i++;

if(T())
```

```
{
if(EP())
return(1);
else
return(0);
}
else
return(0);
}
else
return(1);
}
T()
{
if(F())
{
if(TP())
return(1);
else
return(0);
}
else
return(0);
}
TP()
{
if(input[i]=='*')
```

```
{
i++;
if(F())
{
if(TP())
return(1);
else
return(0);
}
else
return(0);
}
else
return(1);
}
F()
{
if(input[i]=='(')
{
i++;
if(E())
{
if(input[i]==')')
{
i++;
return(1);
}

else

return(0);

}

else

return(0);

}

else if(input[i]>='a'&&input[i]<='z'||
input[i]>='A'&&input[i]<='Z')

{

i++;

return(1);

}

else

return(0);

}
```

INPUT & OUTPUT:

```
Recursive descent parsing for the following grammar

E->TE'

E'->+TE'/@

T->FT'

T'->*FT'/@

F->(E)/ID

Enter the string to be checked:(a+b)*c

String is accepted

Recursive descent parsing for the following grammar

E->TE'

E'->+TE'/@
```

```
T->FT'

T'->*FT'/@

F->(E)/id

Enter the string to be checked:a/c+d

String is not accepted
```

16. Construct a Shift Reduce Parser for a given language

Program for implementing Shift Reduce Parsing using C

ALGORITHM:

1. Get the input expression and store it in the input buffer.
2. Read the data from the input buffer one at the time.
3. Using stack and push & pop operation shift and reduce symbols with respect to production rules available.
4. Continue the process till symbol shift and production rule reduce reaches the start symbol.
5. Display the Stack Implementation table with corresponding Stack actions with input symbols.

PROGRAM:

```
#include

#include

#include

#include

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

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;
}
```

<u>OUTPUT:</u>

```
 SHIFT REDUCE PARSER
 GRAMMER
 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
```

17. Write a program to perform constant propagation.

ALGORITHM:

```
1. For all 4-tuples do
     1.1 get next 4-tuple
1.2 If the operator is "Label" then
     1.2.1 free the constant table
     1.2.2 write the 4-tuple
1.3 If either or both of the operands are in the constant table then
     1.3.1 substitute the constant value(s) for the operand(s)
     1.3.2 if either of the operands are intermediate results then
          1.3.2.1 remove those operand(s) from the constant table.
1.4 if 4-tuple operand(s) are constant then
     1.4.1 if operator is "JPCT" then
          1.4.1.1 if the operand is true then
               1.4.1.1.1 replace the JPCT with JP
               1.4.1.1.2 write the new 4-tuple else
               1.4.1.1.3 eliminate the 4-tuple
     1.4.2 if operator is "=" then
          1.4.2.1 place the variable and constant in the constant table
          1.4.2.2 write the 4-tuple with the constant operand
     1.4.3 else
          1.4.3.1 Perform the operation.
          1.4.3.2 Place the intermediate result name and constant value in the constant table.
Else
     1.4.4 write the 4-tuple Consider the following sequence.
```

PROGRAM:

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>
#include<conio.h>
void input();
void output();
void change(int p,char *res);
void constant();
struct expr
{
char op[2],op1[5],op2[5],res[5];
int flag;
}arr[10];
int n;
void main()
{
```

```
clrscr();
input();
constant();
output();
getch();
}
void input()
{
int i;
printf("\n\nEnter the maximum number of expressions : ");
scanf("%d",&n);
printf("\nEnter the input : \n");
for(i=0;i<n;i++)
{
scanf("%s",arr[i].op);
scanf("%s",arr[i].op1);
scanf("%s",arr[i].op2);
scanf("%s",arr[i].res);
arr[i].flag=0;
}
}
void constant()
{
int i;
int op1,op2,res;
char op,res1[5];
for(i=0;i<n;i++)
{
if(isdigit(arr[i].op1[0]) && isdigit(arr[i].op2[0]) ||
strcmp(arr[i].op,"=")==0) /*if both digits, store them in
variables*/
{
op1=atoi(arr[i].op1);
op2=atoi(arr[i].op2);
op=arr[i].op[0];
switch(op)
{
case '+':
res=op1+op2;
break;
case '-':
res=op1-op2;
break;
case '*':
res=op1*op2;
break;
case '/':
res=op1/op2;
break;
case '=':
res=op1;
break;
}
sprintf(res1,"%d",res);
arr[i].flag=1; /*eliminate expr and replace any operand below that
uses result of this expr */
change(i,res1);
}
}
}
void output()
{
int i=0;
printf("\nOptimized code is : ");
for(i=0;i<n;i++)
{
if(!arr[i].flag)
{
printf("\n%s %s %s
%s",arr[i].op,arr[i].op1,arr[i].op2,arr[i].res);
}
}
}
void change(int p,char *res)
{
int i;
for(i=p+1;i<n;i++)
{
if(strcmp(arr[p].res,arr[i].op1)==0)
strcpy(arr[i].op1,res);
else if(strcmp(arr[p].res,arr[i].op2)==0)
strcpy(arr[i].op2,res);
}
}
```

INPUT:

```
Enter the maximum number of expressions : 4

Enter the input :
= 3 - a
+ a b t1
+ a c t2
+ t1 t2 t3
```

OUTPUT:

```
Optimized code is :
+ 3 b t1
+ 3 c t2
+ t1 t2 t3
```

18. C program to implement intermediate code generation for simple expression.

Program

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int i=1,j=0,no=0,tmpch=90;
char str[100],left[15],right[15];
void findopr();
void explore();
void fleft(int);
void fright(int);
struct exp
{
int pos;
char op;
}k[15];
void main()
{
printf("\t\tINTERMEDIATE CODE GENERATION\n\n");
printf("Enter the Expression :");
scanf("%s",str);
printf("The intermediate code:\n");
findopr();
explore();
}
void findopr()
{
for(i=0;str[i]!='\0';i++)
if(str[i]==':')
{
k[j].pos=i;
k[j++].op=':';
}
for(i=0;str[i]!='\0';i++)
if(str[i]=='/')
{
k[j].pos=i;
k[j++].op='/';
}
```

```
for(i=0;str[i]!='\0';i++)
if(str[i]=='*')
{
k[j].pos=i;
k[j++].op='*';
}
for(i=0;str[i]!='\0';i++)
if(str[i]=='+')
{
k[j].pos=i;
k[j++].op='+';
}
for(i=0;str[i]!='\0';i++)
if(str[i]=='-')
{
k[j].pos=i;
k[j++].op='-';
}
}
void explore()
{
i=1;
while(k[i].op!='\0')
{
fleft(k[i].pos);
fright(k[i].pos);
str[k[i].pos]=tmpch--;
printf("\t%c := %s%c%s\t\t",str[k[i].pos],left,k[i].op,right);
printf("\n");
i++;
}
fright(-1);
if(no==0)
{
fleft(strlen(str));
printf("\t%s := %s",right,left);
getch();
exit(0);
}
printf("\t%s := %c",right,str[k[--i].pos]);
getch();
}
void fleft(int x)
{
```

```
int w=0,flag=0;
x--;
while(x!= -1 &&str[x]!= '+' &&str[x]!='*'&&str[x]!='='&&str[x]!
='\0'&&str[x]!='-'&&str[x]!='/'&&str[x]!=':')
{
if(str[x]!='$'&& flag==0)
{
left[w++]=str[x];
left[w]='\0';
str[x]='$';
flag=1;
}
x--;
}
}
void fright(int x)
{
int w=0,flag=0;
x++;
while(x!= -1 && str[x]!= '+'&&str[x]!='*'&&str[x]!='\0'&&str[x]!
='='&&str[x]!=':'&&str[x]!='-'&&str[x]!='/')
{
if(str[x]!='$'&& flag==0)
{
right[w++]=str[x];
right[w]='\0';
str[x]='$';
flag=1;
}
x++;
}
}
```

19. 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 ofintermediate code (terminated
byexit):\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();}
```