LEX

1.No.of words lines

.c

```
#include <stdio.h>
 int main()
 {
     int number1, number2, sum;
 printf("Enter two integers: ");
 scanf("%d %d", &number1, &number2);
 sum = number1 + number2;
     printf("%d + %d = %d", number1, number2, sum);
  return 0;
 }
```

.l

```
%{
int nchar, nword, nline;
%}
%%
\n { nline++; nchar++; }
[^ \t\n]+ { nword++, nchar += yyleng; }
. { nchar++; }
%%
int yywrap(void) {
return 1;
}
int main(int argc, char *argv[]) {
yyin = fopen(argv[1], "r");
yylex();
printf("Number of characters = %d\n", nchar);
printf("Number of words = %d\n", nword);
printf("Number of lines = %d\n", nline);
```

```
fclose(yyin);
}
```

2.count constants

.c

```
#define P 314
#include<stdio.h> #include<conio.h>
 void main()
{
        int a,b,c = 30;
printf("hello");
}
```

.l

```
digit [0-9]
%{
int cons=0;
%}
%%
{digit}+ { cons++; printf("%s is a constant\n", yytext);  }
.|\n { }
%%
int yywrap(void) {
return 1; }
int main(void)
{
FILE *f;
char file[10];
printf("Enter File Name : ");
scanf("%s",file);
f = fopen(file,"r");
```

```
yyin = f;
yylex();
printf("Number of Constants : %d\n", cons);
fclose(yyin);
}
```

.txt

```
Lo tgyh 8 52 65 86
```

3.no.of macros

.c

```
#define PI 3.14
#include<stdio.h>
#include<conio.h>
 void main()
{
int a,b,c = 30;
printf("hello");
}
```

.l

```
%{
int nmacro, nheader;
%}
%%
^#define { nmacro++; }
^#include { nheader++; }
.|\n {  }
%%
int yywrap(void) {
return 1;
}
int main(int argc, char *argv[]) {
```

```
yyin = fopen(argv[1], "r");
yylex();
printf("Number of macros defined = %d\n", nmacro);
printf("Number of header files included = %d\n", nheader);
fclose(yyin);
}
```

4.

5.count no.of lines

.c

```
#define PI 3.14
#include<stdio.h>
#include<conio.h>
  void main()
{
int a,b,c = 30;
printf("hello");
}
```

.l

```
%{
int yylineno;
%}
%%
^(.*)\n printf("%4d\t%s", ++yylineno, yytext);
%%
int yywrap(void) {
return 1;
}
int main(int argc, char *argv[]) {
yyin = fopen(argv[1], "r");
yylex();
```

```
fclose(yyin);
}
```

6.email id

```
.l
%{
int flag=0;
%}
%%
[a-z . 0-9]+@[a-z]+".com"|".in" { flag=1; }
%%
int main()
{
yylex();
if(flag==1)
printf("Accepted");
else
printf("Not Accepted");
}
int yywrap()
{ return 1;
 }
```

7.cnt no.of lines

input.c

```
#include<stdio.h>
int main()
{
int a,b,c; /*varible declaration*/ printf(“enter two numbers”); scanf(“%d %d”,&a,&b);
c=a+b;//adding two numbers printf(“sum is %d”,c);
return 0;
```

```
}
```

Op.c

```
#include<stdio.h>
int main()
{
int a,b,c;/*asdfgh*/
printf(“enter two numbers”); scanf(“%d %d”,&a,&b);//fyghk
c=a+b;
return 0;
}
```

.l

```
%{
int com=0;
%}
%s COMMENT
%%
"/*" {BEGIN COMMENT;}
<COMMENT>"*/" {BEGIN 0; com++;}
<COMMENT>\n {com++;}
<COMMENT>. {;}
\/\/.* {; com++;}
.|\n {fprintf(yyout,"%s",yytext);}
%%
void main(int argc, char *argv[])
{
if(argc!=3)
{
printf("usage : a.exe input.c output.c\n");
exit(0);
}
```

```
yyin=fopen(argv[1],"r");
yyout=fopen(argv[2],"w");
yylex();
printf("\n number of comments are = %d\n",com);
}
int yywrap()
{
return 1;
}
```

8.mob no validity

.l

```
%%
[1-9][0-9]{9} {printf("\nMobile Number Valid\n");}
.+ {printf("\nMobile Number Invalid\n");}
%%
int main()
{
printf("\nEnter Mobile Number : ");
yylex();
printf("\n");
return 0;
}
int yywrap()
{ }
```

9.

10.separate tokens

.c

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

```
{
int a,b,c = 30;
printf("hello");
}
```

```
.l
digit [0-9]
letter [A-Za-z]
%{
int count_id,count_key;
%}
%%
(stdio.h|conio.h) { printf("%s is a standard library\n",yytext); }
(include|void|main|printf|int) { printf("%s is a keyword\n",yytext); count_key++; }
{letter}({letter}|{digit})*  { printf("%s is a identifier\n", yytext); count_id++; }
{digit}+  { printf("%s is a number\n", yytext); }
\"(\\.|[^"\\])*\"  { printf("%s is a string literal\n", yytext); }
.|\n {  }
%%
int yywrap(void) {
return 1;
}
int main(int argc, char *argv[]) {
yyin = fopen(argv[1], "r");
yylex();
printf("number of identifiers = %d\n", count_id);
printf("number of keywords = %d\n", count_key);
fclose(yyin);
}
```

11.pos and neg

```
.l
%{
int positive_no = 0, negative_no = 0;
%}
%%
^[-][0-9]+ {
negative_no++;
printf("negative number = %s\n",
yytext);} // negative number

[0-9]+ {positive_no++;
printf("positive number = %s\n",
yytext);} // positive number
%%
int yywrap(){}
int main()


{
yylex();
printf ("number of positive numbers = %d,"
"number of negative numbers = %d\n",
positive_no, negative_no);
return 0;
}
```

12.url

```
.l
%%
((http)|(ftp))s?:\/\/[a-zA-Z0-9](.[a-z])+(.[a-zA-Z0-9+=?]*)* {printf("\nURL Valid\n");}
```

```
.+ {printf("\nURL Invalid\n");}

%%
void main()
{
printf("\nEnter URL : ");
yylex();
printf("\n");
}
int yywrap()
{
}
```

13.DOB

```
.I
%%
((0[1-9])|([1-2][0-9])|(3[0-1]))\/((0[1-9])|(1[0-2]))\/(19[0-9]{2}|2[0-9]{3}) printf("Valid DoB");
.* printf("Invalid DoB");
%%

int main()
{
 yylex();
 return 0;
}
int yywrap()
{}
```

14.frequency

```
.I
%%
```

```
((0[1-9])|([1-2][0-9])|(3[0-1]))\/((0[1-9])|(1[0-2]))\/(19[0-9]{2}|2[0-9]{3}) printf("Valid DoB");
.* printf("Invalid DoB");
%%

int main()
{
 yylex();
 return 0;
}
int yywrap()
{}
```

15.mathematical induction

```
.l
%{
#undef yywrap
#define yywrap() 1
int f1=0,f2=0;
char oper;
float op1=0,op2=0,ans=0;
void eval();
%}

DIGIT [0-9]
NUM {DIGIT}+(\.{DIGIT}+)?
OP [*/+-]

%%

{NUM} {
if(f1==0)
```

```
{
op1=atof(yytext);
f1=1;
}
else if(f2==-1)
{
op2=atof(yytext);
f2=1;
}
if((f1==1) && (f2==1))
{
eval();
f1=0;
f2=0;
}
}

{OP} {
oper=(char) *yytext;
f2=-1;
}

[\n] {
if(f1==1 && f2==1)
{
eval;
f1=0;
f2=0;
}
}
```

```
%%


int main()
{
yylex();
}


void eval()
{
switch(oper)
{
case '+':
ans=op1+op2;
break;
case '-':
ans=op1-op2;
break;
case '*':
ans=op1*op2;
break;
case '/':
if(op2==0)
{
printf("ERROR");
return;
}
else
{
ans=op1/op2;
```

```
}
break;
default:
printf("operation not available");
break;
}
printf("The answer is = %lf",ans);
}
```

16.dit or not

```
.l
%%
[0-9]+ {printf("\nValid digit \n");}
.* printf("\nInvalid digit\n");
%%
int yywrap(){}
int main()


{
yylex();
return 0;
}
```

17.vow and const

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

```
%%
[aeiouAEIOU] {vow_count++;}
[a-zA-Z] {const_count++;}
%%
int yywrap(){}
int main()
{
   printf("Enter the string of vowels and consonants:");
   yylex();
   printf("Number of vowels are:  %d\n", vow_count);
   printf("Number of consonants are:  %d\n", const_count);
   return 0;
}
```

18.lon word

```
.l
%{
#include<stdio.h>
int k=0;
%}
%%
[a-zA-Z]+ {
if(yyleng>k)
{  k= yyleng;
}
}
%%
int yywrap(void)
{
return 1;
}
```

```
int main(int argc[],char **argv[])
{
 yyin=fopen("a.txt","r");
 yylex();
 printf("largest: %d",k);
 printf("\n");
 return 0;
}
```

19.replace a word

```
.l
%%
[A-Z]+[\t\n ] { printf("%s is a capital word\n",yytext); }
. ;
%%

int main( )
{
printf("Enter String :\n");
yylex();
}
int yywrap( )
{
return 1;
}
```

20.cap letters

```
.l
%{
```

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

%%
"saveetha"   { printf("sse"); }
.|\n       { putchar(yytext[0]); }
%%

int yywrap() {
   return 1;  // Indicate end of input
}

int main(int argc, char *argv[]) {
   if (argc != 2) {
      fprintf(stderr, "Usage: %s input_file\n", argv[0]);
      return 1;
   }

   FILE *input_file = fopen(argv[1], "r");
   if (!input_file) {
      perror("Error opening file");
      return 1;
   }

   yyin = input_file;
   yylex();

   fclose(input_file);
   return 0;
}
```

C prgms

1.comment or not

```
#include<stdio.h>
#include<conio.h>
int main()
{
char com[30];
int i=2,a=0;
printf("\n Enter comment:");
gets(com);
if(com[0]=='/'){
if(com[1]=='/')
printf("\n It is a comment");
else if(com[1]=='*')
                                                                                {
for(i=2;i<=30;i++)
{
if(com[i]=='*'&&com[i+1]=='/')
{printf("\n It is a comment");
a=1;
break;
}
else
continue;
}
if(a==0)
printf("\n It is not a comment");
}
else
printf("\n It is not a comment");
}
```

```
else
printf("\n It is not a comment");
}
```

2.identifiers constants

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
int main()
        {
int i,ic=0,m,cc=0,oc=0,j;
char b[30],operators[30],identifiers[30],constants[30];
printf("enter the string : ");
scanf("%[^\n]s",&b);
for(i=0;i<strlen(b);i++)
{
if(isspace(b[i]))
{
continue;
}
else if(isalpha(b[i]))
{
identifiers[ic] =b[i];
ic++;
}
else if(isdigit(b[i]))
{
m=(b[i]-'0');
i=i+1;
while(isdigit(b[i]))
{
m=m*10 + (b[i]-'0');
i++;
}
i=i-1;
constants[cc]=m;
```

```
cc++;
}
else
{
if(b[i]=='*')
{
operators[oc]='*';
oc++;
}
else if(b[i]=='-')
{
operators[oc]='-';
oc++;
}
else if(b[i]=='+')
{
operators[oc]='+';
oc++;
}
else if(b[i]=='=')
{
operators[oc]='=';
oc++;
}
}
}
   printf(" identifiers : ");
    for(j=0;j<ic;j++)
 {
       printf("%c ",identifiers[j]);
    }
   printf("\n constants : ");
    for(j=0;j<cc;j++)
 {
       printf("%d ",constants[j]);
    }
   printf("\n operators : ");
     for(j=0;j<oc;j++)
```

```
  {
      printf("%c ",operators[j]);
    }
}
```

3.ignore redundant

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

int isKeyword(char buffer[]){
char keywords[32][10] = {"main","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",
"unsigned","void","printf","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("flex_input.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(ch == '0'||ch =='1'||ch =='2'||ch =='3'||ch =='4'||ch =='6'||ch =='7'||ch =='8'||ch =='9'){
```

```
  printf("%d is number\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 identifier\n", buffer);
  }


}
fclose(fp);
return 0;
}
```

4.follow

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
int limit, x = 0;
char production[10][10], array[10];


void find_first(char ch);
void find_follow(char ch);
void Array_Manipulation(char ch);


int main()
{
    int count;
    char option, ch;
    printf("\nEnter Total Number of Productions:\t");
    scanf("%d", &limit);
    for(count = 0; count < limit; count++)
    {
        printf("\nValue of Production Number [%d]:\t", count + 1);
        scanf("%s", production[count]);
```

```
    }
    do
    {
        x = 0;
        printf("\nEnter production Value to Find Follow:\t");
        scanf(" %c", &ch);
        find_follow(ch);
        printf("\nFollow Value of %c:\t{ ", ch);
        for(count = 0; count < x; count++)
        {
            printf("%c ", array[count]);
        }
        printf("}\n");
        printf("To Continue, Press Y:\t");
        scanf(" %c", &option);
    }while(option == 'y' || option == 'Y');
    return 0;
}

void find_follow(char ch)
{
    int i, j;
    int length = strlen(production[i]);
    if(production[0][0] == ch)
    {
        Array_Manipulation('$');
    }
    for(i = 0; i < limit; i++)
    {
        for(j = 2; j < length; j++)
        {
            if(production[i][j] == ch)
            {
                if(production[i][j + 1] != '\0')
                {
                    find_first(production[i][j + 1]);
                }
                if(production[i][j + 1] == '\0' && ch != production[i][0])
                {
                    find_follow(production[i][0]);
                }
            }
        }
    }
```

```
}


void find_first(char ch)
{
    int i, k;
    if(!(isupper(ch)))
    {
        Array_Manipulation(ch);
    }
    for(k = 0; k < limit; k++)
    {
        if(production[k][0] == ch)
        {
            if(production[k][2] == '$')
            {
                find_follow(production[i][0]);
            }
            else if(islower(production[k][2]))
            {
                Array_Manipulation(production[k][2]);
            }
            else
            {
                find_first(production[k][2]);
            }
        }
    }
}


void Array_Manipulation(char ch)
{
    int count;
    for(count = 0; count <= x; count++)
    {
        if(array[count] == ch)
        {
            return;
        }
    }
    array[x++] = ch;
}
```

5.symb of tab operator

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

```
#include<stdlib.h>
#include<string.h>
int cnt=0;
struct symtab
        {
char label[20];
int addr;
        }
        sy[50];
        void insert();
        int search(char *);
        void display();
        void modify();
        int main()
        {
        int ch,val;
        char lab[10];
        do
        {
printf("\n1.insert\n2.display\n3.search\n4.modify\n5.exit\n");
scanf("%d",&ch);
switch(ch)
{
case 1:
insert();
        break;
case 2:
display();
break;
case 3:
printf("enter the label");
scanf("%s",lab);
val=search(lab);
if(val==1)
printf("label is found");
else
printf("label is not found");
break;
case 4:
modify();
break;
case 5:
exit(0);
break;
```

```
}
}while(ch<5);
}
void insert()
        {
int val;
char lab[10];
int symbol;
printf("enter the label");
scanf("%s",lab);
val=search(lab);
if(val==1)
printf("duplicate symbol");
else
{
strcpy(sy[cnt].label,lab);
printf("enter the address");
scanf("%d",&sy[cnt].addr);
cnt++;
}
}
int search(char *s)
{
int flag=0,i; for(i=0;i<cnt;i++)
{
if(strcmp(sy[i].label,s)==0)
flag=1;
}
return flag;
}
void modify()
{
int val,ad,i;
char lab[10];
printf("enter the labe:");
scanf("%s",lab);
val=search(lab);
if(val==0)
printf("no such symbol");
else
{
printf("label is found \n");
printf("enter the address");
scanf("%d",&ad);
```

```
for(i=0;i<cnt;i++)
{
if(strcmp(sy[i].label,lab)==0)
sy[i].addr=ad;
}
}
          }
void display()
{
int i;
for(i=0;i<cnt;i++)
printf("%s\t%d\n",sy[i].label,sy[i].addr);
}
```

6.first

```
#include<stdio.h>
#include<ctype.h>
void FIRST(char[],char );
void addToResultSet(char[],char);
int numOfProductions;
char productionSet[10][10];
int main()
{
  int i;
  char choice;
  char c;
  char result[20];
  printf("How many number of productions ? :");
  scanf(" %d",&numOfProductions);
  for(i=0;i<numOfProductions;i++)//read production string eg: E=E+T
  {
    printf("Enter productions Number %d : ",i+1);
    scanf(" %s",productionSet[i]);
  }
  do
  {
    printf("\n Find the FIRST of  :");
    scanf(" %c",&c);
    FIRST(result,c); //Compute FIRST; Get Answer in 'result' array
    printf("\n FIRST(%c)= { ",c);
    for(i=0;result[i]!='\0';i++)
    printf(" %c ",result[i]);      //Display result
    printf("}\n");
```

```
        printf("press 'y' to continue : ");
       scanf(" %c",&choice);
    }
    while(choice=='y'||choice =='Y');
}
/*
 *Function FIRST:
 *Compute the elements in FIRST(c) and write them
 *in Result Array.
 */
void FIRST(char* Result,char c)
{
    int i,j,k;
    char subResult[20];
    int foundEpsilon;
    subResult[0]='\0';
    Result[0]='\0';
    //If X is terminal, FIRST(X) = {X}.
    if(!(isupper(c)))
    {
       addToResultSet(Result,c);
            return ;
    }
    //If X is non terminal
    //Read each production
    for(i=0;i<numOfProductions;i++)
    {
//Find production with X as LHS
       if(productionSet[i][0]==c)
       {
//If X → ε is a production, then add ε to FIRST(X).
 if(productionSet[i][2]=='$') addToResultSet(Result,'$');
           //If X is a non-terminal, and X → Y1 Y2 … Yk
           //is a production, then add a to FIRST(X)
           //if for some i, a is in FIRST(Yi),
           //and ε is in all of FIRST(Y1), …, FIRST(Yi-1).
      else
           {
              j=2;
              while(productionSet[i][j]!='\0')
              {
              foundEpsilon=0;
              FIRST(subResult,productionSet[i][j]);
              for(k=0;subResult[k]!='\0';k++)
```

```
                addToResultSet(Result,subResult[k]);
              for(k=0;subResult[k]!='\0';k++)
                if(subResult[k]=='$')
                {
                   foundEpsilon=1;
                   break;
                }

              if(!foundEpsilon)
                break;
              j++;
             }
          }
    }
}
   return ;
}

void addToResultSet(char Result[],char val)
{
   int k;
   for(k=0 ;Result[k]!='\0';k++)
      if(Result[k]==val)
         return;
   Result[k]=val;
   Result[k+1]='\0';
}
```

7. elimination of L recursion

```
#include<stdio.h>
#include<string.h>
#define SIZE 10
  int main () {
      char non_terminal;
      char beta,alpha;
      int num;
      char production[10][SIZE];
      int index=3; /* starting of the string following "->" */
      printf("Enter Number of Production : ");
      scanf("%d",&num);
      printf("Enter the grammar as E->E-A :\n");
```

```
    for(int i=0;i<num;i++){
        scanf("%s",production[i]);
    }
    for(int i=0;i<num;i++){
        printf("\nGRAMMAR : : : %s",production[i]);
        non_terminal=production[i][0];
        if(non_terminal==production[i][index]) {
            alpha=production[i][index+1];
            printf(" is left recursive.\n");
            while(production[i][index]!=0 && production[i][index]!='|')
                index++;
            if(production[i][index]!=0) {
                beta=production[i][index+1];
                printf("Grammar without left recursion:\n");
                printf("%c->%c%c\'",non_terminal,beta,non_terminal);
                printf("\n%c\'->%c%c\'|E\n",non_terminal,alpha,non_terminal);
            }
            else
                printf(" can't be reduced\n");
        }
        else
            printf(" is not left recursive.\n");
        index=3;
    }
}
```

8.Recursive decent parsing

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

// Function declarations
void error();
void E();
void Eprime();
void T();
void Tprime();
void F();
```

```
void match(char token);q

// Global variables
char input[100];  // Input string
int pos = 0;     // Position of the current token

int main() {
   printf("Enter an expression: ");
   fgets(input, sizeof(input), stdin);
   input[strcspn(input, "\n")] = '\0'; // Remove newline character

   E();  // Start parsing

   if (input[pos] == '\0') {
      printf("Parsing successful.\n");
   } else {
      printf("Parsing failed.\n");
   }

   return 0;
}

void error() {
   printf("Parsing error.\n");
   exit(1);
}

void match(char token) {
   if (input[pos] == token) {
      pos++;
   } else {
      error();
   }
}

void E() {
   T();
   Eprime();
}

void Eprime() {
   if (input[pos] == '+') {
      match('+');
      T();
```

```
    Eprime();
  }
}

void T() {
  F();
  Tprime();
}

void Tprime() {
  if (input[pos] == '*') {
    match('*');
    F();
    Tprime();
  }
}

void F() {
  if (input[pos] == '(') {
    match('(');
    E();
    match(')');
  } else if (isalnum(input[pos])) {
    pos++;
  } else {
    error();
  }
}
```

9.shift parsing

```
#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();
int 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;
}
```

10.Quadtple

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct {
    char op[10];
    char arg1[10];
    char arg2[10];
    char result[10];
} Quadruple;

void generateThreeAddressCode(const char *expression, Quadruple *quadruples, int *quadCount) {

    char temp[10];
```

```
    int tempCounter = 0;
    char operatorStack[10];
    int operatorTop = -1;

    for (int i = 0; i < strlen(expression); i++) {
        if (expression[i] >= 'a' && expression[i] <= 'z') {
            temp[tempCounter++] = expression[i];
            temp[tempCounter] = '\0';
        } else if (expression[i] == '+' || expression[i] == '-' || expression[i] == '*' || expression[i] == '/') {
            if (operatorTop != -1) {
                if (operatorStack[operatorTop] == '*' || operatorStack[operatorTop] == '/') {
                    sprintf(quadruples[*quadCount].op, "%c", operatorStack[operatorTop--]);
                    sprintf(quadruples[*quadCount].arg1, "%s", temp);
                    sprintf(quadruples[*quadCount].arg2, "%c", '\0');
                    sprintf(quadruples[*quadCount].result, "T%d", (*quadCount)+1);
                    (*quadCount)++;
                }
            }
            operatorStack[++operatorTop] = expression[i];
            tempCounter = 0;
        }
    }

    for (int j = operatorTop; j >= 0; j--) {
        sprintf(quadruples[*quadCount].op, "%c", operatorStack[j]);
        sprintf(quadruples[*quadCount].arg1, "%s", temp);
        sprintf(quadruples[*quadCount].arg2, "%c", '\0');
        sprintf(quadruples[*quadCount].result, "T%d", (*quadCount)+1);
        (*quadCount)++;
    }
}

void printQuadruples(const Quadruple *quadruples, int quadCount) {
```

```
    printf("Quadruples:\n");
    for (int i = 0; i < quadCount; i++) {
        printf("(%s, %s, %s, %s)\n", quadruples[i].op, quadruples[i].arg1, quadruples[i].arg2, quadruples[i].result);
    }
}

int main() {
    char expression[100];
    printf("Enter an arithmetic expression: ");
    scanf("%s", expression);

    Quadruple quadruples[100];
    int quadCount = 0;

    generateThreeAddressCode(expression, quadruples, &quadCount);
    printQuadruples(quadruples, quadCount);

    return 0;
}
```

11.Triple

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

typedef struct {
    char op;
    char arg1[10];
    char arg2[10];
    char result[10];
} Triple;

Triple triples[50];
```

```
int nextTriple = 0;

void generateTriple(char op, char arg1[], char arg2[], char result[]) {
    Triple newTriple;
    newTriple.op = op;
    strcpy(newTriple.arg1, arg1);
    strcpy(newTriple.arg2, arg2);
    strcpy(newTriple.result, result);
    triples[nextTriple++] = newTriple;
}

int main() {
    char expr[100];
    printf("Enter a simple expression: ");
    scanf("%s", expr);

    char temp[10];
    int tempCount = 0;
    for (int i = 0; i < strlen(expr); i++) {
        if (expr[i] == '+' || expr[i] == '-' || expr[i] == '*' || expr[i] == '/') {
            temp[tempCount] = '\0';
            generateTriple(expr[i], temp, "", "");
            tempCount = 0;
        } else {
            temp[tempCount++] = expr[i];
        }
    }
    temp[tempCount] = '\0';
    generateTriple('=', temp, "", "result");

    printf("Generated Triples:\n");
    printf("Op\tArg1\tArg2\tResult\n");
    for (int i = 0; i < nextTriple; i++) {
```

```
        printf("%c\t%s\t%s\t%s\n", triples[i].op, triples[i].arg1, triples[i].arg2, triples[i].result);
    }

    return 0;
}
```

12.left factoring

```
#include<stdio.h>
#include<string.h>
  int main()
  {
      char gram[20],part1[20],part2[20],modifiedGram[20],newGram[20],tempGram[20];
      int i,j=0,k=0,l=0,pos;
      printf("Enter Production : S->");
      gets(gram);
      for(i=0;gram[i]!='|';i++,j++)
          part1[j]=gram[i];
      part1[j]='\0';
      for(j=++i,i=0;gram[j]!='\0';j++,i++)
          part2[i]=gram[j];
      part2[i]='\0';
      for(i=0;i<strlen(part1)||i<strlen(part2);i++)
      {
          if(part1[i]==part2[i])
          {
              modifiedGram[k]=part1[i];
              k++;
              pos=i+1;
          }
      }
      for(i=pos,j=0;part1[i]!='\0';i++,j++){
          newGram[j]=part1[i];
```

```
    }
    newGram[j++]='|';
    for(i=pos;part2[i]!='\0';i++,j++){
        newGram[j]=part2[i];
    }
    modifiedGram[k]='X';
    modifiedGram[++k]='\0';
    newGram[j]='\0';
    printf("\n S->%s",modifiedGram);
    printf("\n X->%s\n",newGram);
 }
```

13.backend

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
int n,i,j;
char a[50][50];
printf("enter the no: intermediate code:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter the 3 address code:%d:",i+1);
for(j=0;j<6;j++)
{
scanf("%c",&a[i][j]);
}
}
printf("the generated code is:");
for(i=0;i<n;i++)
{
```

```
printf("\n mov %c,R%d",a[i][3],i);
if(a[i][4]=='-')
{
printf("\n sub %c,R%d",a[i][5],i);
}
if(a[i][4]=='+')
{
printf("\n add %c,R%d",a[i][5],i);
}
if(a[i][4]=='*')
{
printf("\n mul %c,R%d",a[i][5],i);
}
if(a[i][4]=='/')
{
printf("\n div %c,R%d",a[i][5],i);
}
printf("\n mov R%d,%c",i,a[i][1]);
printf("\n");
}
return 0;
}
```

14.newlines,whitespaces

```
#include <stdio.h>
int main()
{
   char str[100];
   int words=0,newline=0,characters=0;
   scanf("%[^~]",&str);
   for(int i=0;str[i]!='\0';i++)
    {
       if(str[i] == ' ')
```

```
        {
            words++;
        }
        else if(str[i] == '\n')
        {
           newline++;
            words++;
        }
        else if(str[i] != ' ' && str[i] != '\n'){
        characters++;
        }
    }
   if(characters > 0)
   {
      words++;
      newline++;
   }
    printf("Total number of words : %d\n",words);
    printf("Total number of lines : %d\n",newline);
    printf("Total number of characters : %d\n",characters);
   return 0;
}
```

15.op precendence

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

#define STACK_SIZE 50

char stack[STACK_SIZE];
int top = -1;
void push(char item) {
```

```
    if (top >= STACK_SIZE - 1) {
        printf("Stack Overflow!\n");
        exit(1);
    }
    stack[++top] = item;
}

char pop() {
    if (top == -1) {
        printf("Stack Underflow!\n");
        exit(1);
    }
    return stack[top--];
}
int precedence(char op) {
    if (op == '+' || op == '-')
        return 1;
    else if (op == '*' || op == '/')
        return 2;
    return 0;
}

void operatorPrecedenceParsing(char input[]) {
    int i = 0;

    printf("Stack\tInput\tAction\n");

    while (input[i] != '\0') {
        if (input[i] == '(') {
            push(input[i]);
            i++;
        } else if (isalnum(input[i])) {
            printf("%s\t", stack);
            printf("%s\t", input + i);
            printf("Shift %c\n", input[i]);
            push(input[i]);
            i++;
        } else if (input[i] == ')') {
```

```
            while (top != -1 && stack[top] != '(') {
                printf("%s\t", stack);
                printf("%s\t", input + i);
                printf("Reduce %c\n", pop());
            }
            if (top != -1 && stack[top] == '(') {
                pop();
            }
            i++;
        } else {
            while (top != -1 && precedence(stack[top]) >= precedence(input[i])) {
                printf("%s\t", stack);
                printf("%s\t", input + i);
                printf("Reduce %c\n", pop());
            }
            push(input[i]);
            i++;
        }
    }

    while (top != -1) {
        printf("%s\t", stack);
        printf(" \t");
        printf("Reduce %c\n", pop());
    }

    printf("ACCEPTED\n");
}

int main() {
    char input[50];

    printf("Enter an arithmetic expression: ");
    scanf("%s", input);

    operatorPrecedenceParsing(input);

    return 0;
```

```
}
```

16.elim coma,subexp

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

typedef struct {
   char op;
   char arg1[10];
   char arg2[10];
   char result[10];
} Triple;

Triple triples[50];
int nextTriple = 0;

void generateTriple(char op, char arg1[], char arg2[], char result[]) {
   Triple newTriple;
   newTriple.op = op;
   strcpy(newTriple.arg1, arg1);
   strcpy(newTriple.arg2, arg2);
   strcpy(newTriple.result, result);
   triples[nextTriple++] = newTriple;
}

int findCommonSubexpression(char op, char arg1[], char arg2[]) {
   for (int i = 0; i < nextTriple; i++) {
      if (triples[i].op == op && strcmp(triples[i].arg1, arg1) == 0 && strcmp(triples[i].arg2, arg2) == 0) {
         return i;
      }
   }
   return -1;
}

int main() {
   generateTriple('+', "a", "b", "t1");
   generateTriple('-', "t1", "c", "t2");
   generateTriple('*', "a", "b", "t3");
   generateTriple('+', "t2", "t3", "t4");
```

```
    generateTriple('/', "t4", "t1", "result");
    generateTriple('+', "a", "b", "t5");
    generateTriple('-', "t5", "c", "t6");
    generateTriple('*', "a", "b", "t7");
    generateTriple('+', "t6", "t7", "t8");
    generateTriple('/', "t8", "t5", "result");

    printf("Original Triples:\n");
    printf("Op\tArg1\tArg2\tResult\n");
    for (int i = 0; i < nextTriple; i++) {
        printf("%c\t%s\t%s\t%s\n", triples[i].op, triples[i].arg1, triples[i].arg2, triples[i].result);
    }

    int changed = 0;
    for (int i = 0; i < nextTriple; i++) {
        int commonIndex = findCommonSubexpression(triples[i].op, triples[i].arg1, triples[i].arg2);
        if (commonIndex != -1 && commonIndex < i) {
            strcpy(triples[i].result, triples[commonIndex].result);
            changed = 1;
        }
    }

    if (changed) {
        printf("\nAfter Common Subexpression Elimination:\n");
        printf("Op\tArg1\tArg2\tResult\n");
        for (int i = 0; i < nextTriple; i++) {
            printf("%c\t%s\t%s\t%s\n", triples[i].op, triples[i].arg1, triples[i].arg2, triples[i].result);
        }
    } else {
        printf("\nNo common subexpressions eliminated.\n");
    }

    return 0;
}
```

17.elim deadcode

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

```
typedef struct {
    char statement[50];
    int isAlive;
} Statement;

Statement statements[50];
int nextStatement = 0;

void addStatement(char statement[]) {
    Statement newStatement;
    strcpy(newStatement.statement, statement);
    newStatement.isAlive = 1;
    statements[nextStatement++] = newStatement;
}

int main() {
    addStatement("x = 5;");
    addStatement("y = x + 3;");
    addStatement("z = x * y;");
    addStatement("x = z - 2;");
    addStatement("y = 10;");

    printf("Original Statements:\n");
    for (int i = 0; i < nextStatement; i++) {
        printf("%s\n", statements[i].statement);
    }

    // Identify and eliminate dead code
    for (int i = nextStatement - 1; i >= 0; i--) {
        if (!strstr(statements[i].statement, "=")) {
            statements[i].isAlive = 0;
        } else {
            char* var = strtok(statements[i].statement, " =;");
            while (var != NULL) {
                for (int j = i + 1; j < nextStatement; j++) {
                    if (strstr(statements[j].statement, var)) {
                        statements[i].isAlive = 1;
```

```
                    break;
                }
            }
            var = strtok(NULL, " =;");
        }
    }
}

printf("\nAfter Dead Code Elimination:\n");
for (int i = 0; i < nextStatement; i++) {
    if (statements[i].isAlive) {
        printf("%s\n", statements[i].statement);
    }
}

return 0;
}
```

18.

19.slr

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

int i,j,k,m,n=0,o,p,ns=0,tn=0,rr=0,ch=0;
char read[15][10],gl[15],gr[15][10],temp,templ[15],tempr[15][10],*ptr,temp2[5],dfa[15][15];

struct states
{
    char lhs[15],rhs[15][10];
    int n;
}I[15];

int compstruct(struct states s1,struct states s2)
{
    int t;
    if(s1.n!=s2.n)
        return 0;
    if( strcmp(s1.lhs,s2.lhs)!=0 )
```

```
        return 0;
    for(t=0;t<s1.n;t++)
        if( strcmp(s1.rhs[t],s2.rhs[t])!=0 )
            return 0;
    return 1;
}

void moreprod()
{
    int r,s,t,l1=0,rr1=0;
    char *ptr1,read1[15][10];

    for(r=0;r<I[ns].n;r++)
    {
        ptr1=strchr(I[ns].rhs[l1],'.');
        t=ptr1-I[ns].rhs[l1];
        if( t+1==strlen(I[ns].rhs[l1]) )
        {
            l1++;
            continue;
        }
        temp=I[ns].rhs[l1][t+1];
        l1++;
        for(s=0;s<rr1;s++)
            if( temp==read1[s][0] )
                break;
        if(s==rr1)
        {
            read1[rr1][0]=temp;
            rr1++;
        }
        else
            continue;

        for(s=0;s<n;s++)
        {
            if(gl[s]==temp)
            {
```

```
                I[ns].rhs[I[ns].n][0]='.';
                I[ns].rhs[I[ns].n][1]=NULL;
                strcat(I[ns].rhs[I[ns].n],gr[s]);
                I[ns].lhs[I[ns].n]=gl[s];
                I[ns].lhs[I[ns].n+1]=NULL;
                I[ns].n++;
            }
        }
    }
}

void canonical(int l)
{
    int t1;
    char read1[15][10],rr1=0,*ptr1;
    for(i=0;i<I[l].n;i++)
    {
        temp2[0]='.';
        ptr1=strchr(I[l].rhs[i],'.');
        t1=ptr1-I[l].rhs[i];
        if( t1+1==strlen(I[l].rhs[i]) )
            continue;

        temp2[1]=I[l].rhs[i][t1+1];
        temp2[2]=NULL;

        for(j=0;j<rr1;j++)
            if( strcmp(temp2,read1[j])==0 )
                break;
        if(j==rr1)
        {
            strcpy(read1[rr1],temp2);
            read1[rr1][2]=NULL;
            rr1++;
        }
        else
            continue;
```

```
for(j=0;j<I[0].n;j++)
{
   ptr=strstr(I[l].rhs[j],temp2);
   if( ptr )
   {
      templ[tn]=I[l].lhs[j];
      templ[tn+1]=NULL;
      strcpy(tempr[tn],I[l].rhs[j]);
      tn++;
   }
}

for(j=0;j<tn;j++)
{
   ptr=strchr(tempr[j],'.');
   p=ptr-tempr[j];
   tempr[j][p]=tempr[j][p+1];
   tempr[j][p+1]='.';
   I[ns].lhs[I[ns].n]=templ[j];
   I[ns].lhs[I[ns].n+1]=NULL;
   strcpy(I[ns].rhs[I[ns].n],tempr[j]);
   I[ns].n++;
}

moreprod();
for(j=0;j<ns;j++)
{
   //if ( memcmp(&I[ns],&I[j],sizeof(struct states))==1 )
   if( compstruct(I[ns],I[j])==1 )
   {
      I[ns].lhs[0]=NULL;
      for(k=0;k<I[ns].n;k++)
         I[ns].rhs[k][0]=NULL;
      I[ns].n=0;
      dfa[l][j]=temp2[1];
      break;
   }
}
```

```
        if(j<ns)
        {
           tn=0;
           for(j=0;j<15;j++)
           {
              templ[j]=NULL;
              tempr[j][0]=NULL;
           }
           continue;
        }

        dfa[l][j]=temp2[1];
        printf("\n\nI%d :",ns);
        for(j=0;j<I[ns].n;j++)
           printf("\n\t%c -> %s",I[ns].lhs[j],I[ns].rhs[j]);

        ns++;
        tn=0;
        for(j=0;j<15;j++)
        {
           templ[j]=NULL;
           tempr[j][0]=NULL;
        }
     }
  }

int main()
{
   FILE *f;
   int l;
   //clrscr();

   for(i=0;i<15;i++)
   {
      I[i].n=0;
      I[i].lhs[0]=NULL;
      I[i].rhs[0][0]=NULL;
      dfa[i][0]=NULL;
```

```
}

f=fopen("slr.txt","r");
while(!feof(f))
{
   fscanf(f,"%c",&gl[n]);
   fscanf(f,"%s\n",gr[n]);
   n++;
}

printf("THE GRAMMAR IS AS FOLLOWS\n");
for(i=0;i<n;i++)
   printf("\t\t\t\t%c -> %s\n",gl[i],gr[i]);

I[0].lhs[0]='Z';
strcpy(I[0].rhs[0],".S");
I[0].n++;
l=0;
for(i=0;i<n;i++)
{
   temp=I[0].rhs[l][1];
   l++;
   for(j=0;j<rr;j++)
      if( temp==read[j][0] )
         break;
   if(j==rr)
   {
      read[rr][0]=temp;
      rr++;
   }
   else
      continue;
   for(j=0;j<n;j++)
   {
      if(gl[j]==temp)
      {
         I[0].rhs[I[0].n][0]='.';
         strcat(I[0].rhs[I[0].n],gr[j]);
```

```
            I[0].lhs[I[0].n]=gl[j];
            I[0].n++;
          }
        }
    }
    ns++;

    printf("\nI%d :\n",ns-1);
    for(i=0;i<I[0].n;i++)
        printf("\t%c -> %s\n",I[0].lhs[i],I[0].rhs[i]);

    for(l=0;l<ns;l++)
        canonical(l);

    printf("\n\n\t\tPRESS ANY KEY FOR DFA TABLE");

    //clrscr();

    printf("\t\t\tDFA TABLE IS AS FOLLOWS\n\n\n");
    for(i=0;i<ns;i++)
    {
        printf("I%d : ",i);
        for(j=0;j<ns;j++)
            if(dfa[i][j]!='\0')
                printf("'%c'->I%d | ",dfa[i][j],j);
        printf("\n\n\n");
    }
    printf("\n\n\n\t\tPRESS ANY KEY TO EXIT");
}
```

20.ambigous

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

#define MAX_PRODUCTIONS 10
#define MAX_SYMBOLS 10
```

```
typedef struct {
   char lhs;
   char rhs[MAX_SYMBOLS];
} Production;

Production grammar[MAX_PRODUCTIONS];
int numProductions = 0;

bool isAmbiguous(const char *input) {
   return true;
}

int main() {
   printf("Enter the number of productions: ");
   scanf("%d", &numProductions);

   printf("Enter the productions in the form A -> XYZ (no spaces): \n");
   for (int i = 0; i < numProductions; i++) {
      scanf(" %c -> %s", &grammar[i].lhs, grammar[i].rhs);
   }

   char input[100];
   printf("Enter a string: ");
   scanf("%s", input);

   if (isAmbiguous(input)) {
      printf("The grammar is ambiguous for the given string.\n");
   } else {
      printf("The grammar is not ambiguous for the given string.\n");
   }

   return 0;
}
```