GIDC Degree Engineering College
(Managed by GIDC Education Society)
BE -IV SEM-VII, ACADEMIC YEAR:2022-23

COMPILER DESIGN (3170701)

PRACTICAL INDEX

PRACTICAL 1

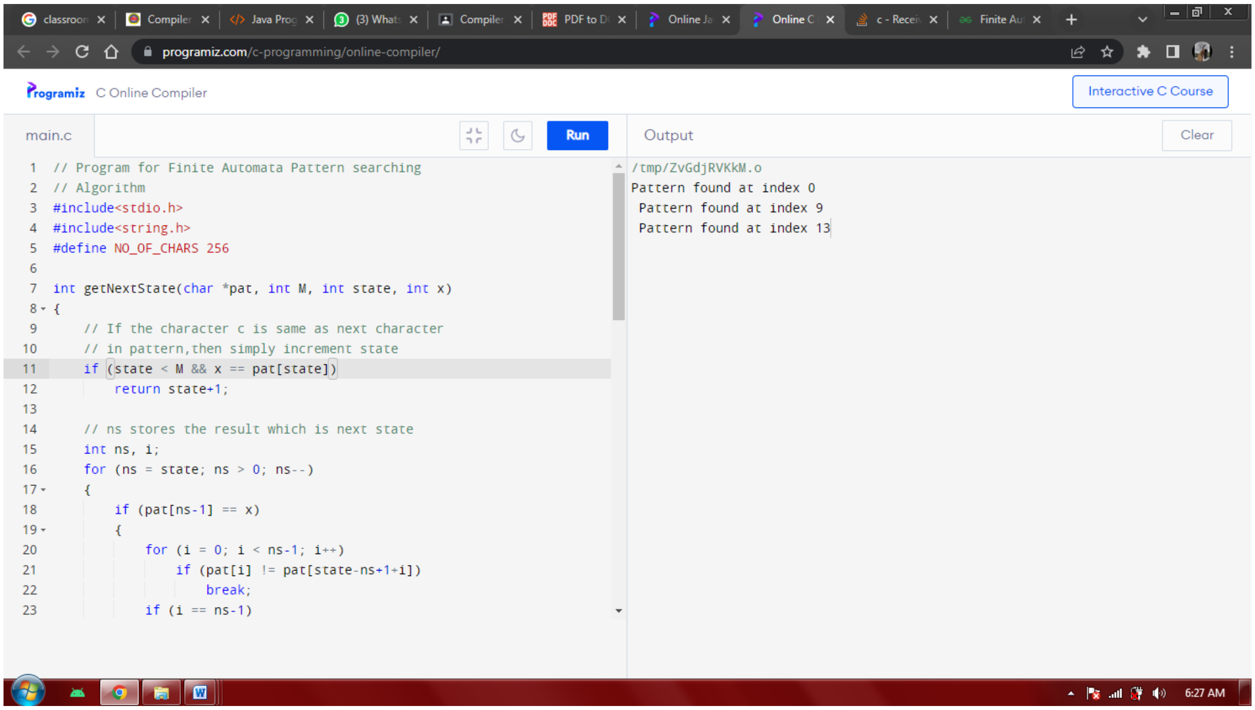
AIM: Implementation of Finite Automata and String Validation

CODE:

```
// Program for Finite Automata Pattern searching
#include<stdio.h>
#include<string.h>
#define NO_OF_CHARS 256

int getNextState(char *pat, int M, int state, int x)
{
    // If the character c is same as next character
    // in pattern,then simply increment state
    if (state < M && x == pat[state])
            return state+1;

    // ns stores the result which is next state
    int ns, i;
    for (ns = state; ns > 0; ns--)
    {
            if (pat[ns-1] == x)
            {
                    for (i = 0; i < ns-1; i++)
                            if (pat[i] != pat[state-ns+1+i])
                                    break;
                    if (i == ns-1)
                            return ns;
            }
    }

    return 0;
}

void computeTF(char *pat, int M, int TF[][NO_OF_CHARS])
{
    int state, x;
    for (state = 0; state <= M; ++state)
            for (x = 0; x < NO_OF_CHARS; ++x)
                    TF[state][x] = getNextState(pat, M, state, x);
}

void search(char *pat, char *txt)
{
    int M = strlen(pat);
    int N = strlen(txt);

    int TF[M+1][NO_OF_CHARS];

    computeTF(pat, M, TF);
```

```
        // Process txt over FA.
        int i, state=0;
        for (i = 0; i < N; i++)
        {
                state = TF[state][txt[i]];
                if (state == M)
                        printf ("\n Pattern found at index %d",i-M+1);
        }
    }

    // Driver program to test above function
    int main()
    {
        char *txt = "AABAACAADAABAAABAA";
        char *pat = "AABA";
        search(pat, txt);
        return 0;
    }
```

OUTPUT:



programiz.com/c-programming/online-compiler/
Programiz C Online Compiler
Interactive C Course
main.c
Run
Output
Clear

```
// Program for Finite Automata Pattern searching
// Algorithm
#include<stdio.h>
#include<string.h>
#define NO_OF_CHARS 256

int getNextState(char *pat, int M, int state, int x)
{
    // If the character c is same as next character
    // in pattern,then simply increment state
    if (state < M && x == pat[state])
        return state+1;

    // ns stores the result which is next state
    int ns, i;
    for (ns = state; ns > 0; ns--)
    {
        if (pat[ns-1] == x)
        {
            for (i = 0; i < ns-1; i++)
                if (pat[i] != pat[state-ns+1+i])
                    break;
            if (i == ns-1)
```

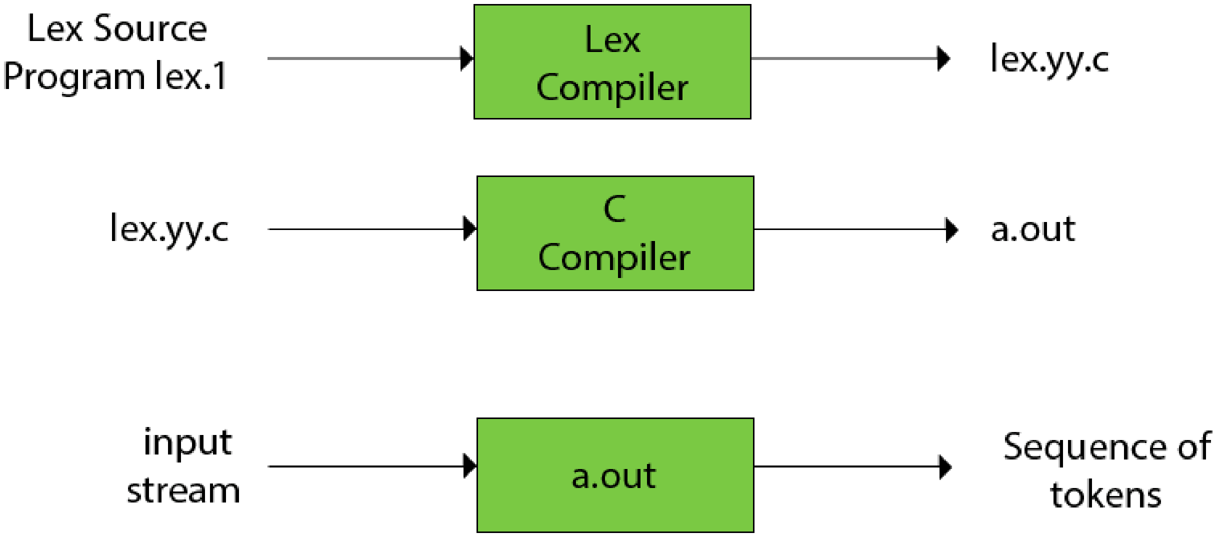
```
/tmp/ZvGdjRVKkM.o
Pattern found at index 0
 Pattern found at index 9
 Pattern found at index 13
```

PRACTICAL 2

AIM: Introduction to Lex Tool.

LEX

- Lex is a program that generates lexical analyzer. It is used with YACC parser generator.
- The lexical analyzer is a program that transforms an input stream into a sequence of tokens.
- It reads the input stream and produces the source code as output through implementing the lexical analyzer in the C program.
- **The function of Lex is as follows:**
- Firstly lexical analyzer creates a program lex.1 in the Lex language. Then Lex compiler runs the lex.1 program and produces a C program lex.yy.c.
- Finally C compiler runs the lex.yy.c program and produces an object program a.out.
- a.out is lexical analyzer that transforms an input stream into a sequence of tokens.



- **Lex file format**

A Lex program is separated into three sections by %% delimiters. The formal of Lex source is as follows:
{ definitions }
%%
{ rules }
%%
{ user subroutines }

- **Definitions** include declarations of constant, variable and regular definitions.
- **Rules** define the statement of form p1 {action1} p2 {action2}....pn {action}.
- Where **pi** describes the regular expression and **action1** describes the actions what action the lexical analyzer should take when pattern pi matches a lexeme.

- **User subroutines** are auxiliary procedures needed by the actions. The subroutine can be loaded with the lexical analyzer and compiled separately.

PRACTICAL 3

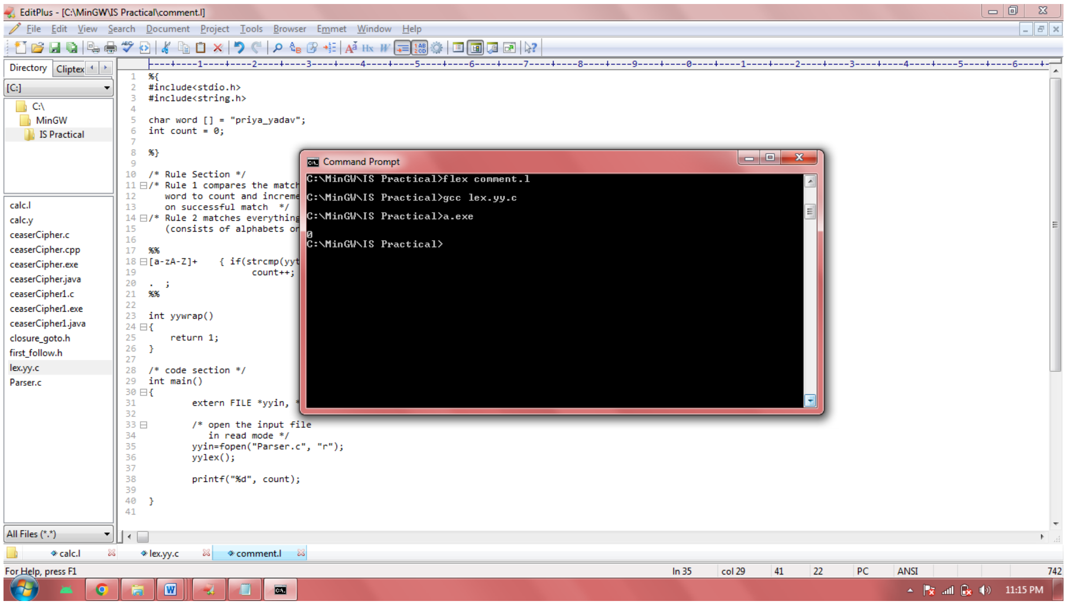
AIM: Implement following Programs Using Lex

a. Generate Histogram of words

Code:

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

char word [] = "priya_yadav";
int count = 0;
%}
%%
[a-zA-Z]+    { if(strcmp(yytext, word)==0)
               count++; }
.  ; %%
int yywrap()
{
   return 1;
}
/* code section */
int main()
{
        extern FILE *yyin, *yyout;
        yyin=fopen("Parser.c", "r");
        yylex();

        printf("%d", count);

}
```

Output



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

char word [] = "priya_yadav";
int count = 0;

%}

/* Rule Section */
/* Rule 1 compares the match
   word to count and increme
   on successful match  */
/* Rule 2 matches everything
   (consists of alphabets or

%%
[a-zA-Z]+    { if(strcmp(yyt
                    count++;
.   ;
%%

int yywrap()
{
    return 1;
}

/* code section */
int main()
{
        extern FILE *yyin, *

        /* open the input file
           in read mode */
        yyin=fopen("Parser.c", "r");
        yylex();

        printf("%d", count);

}
```

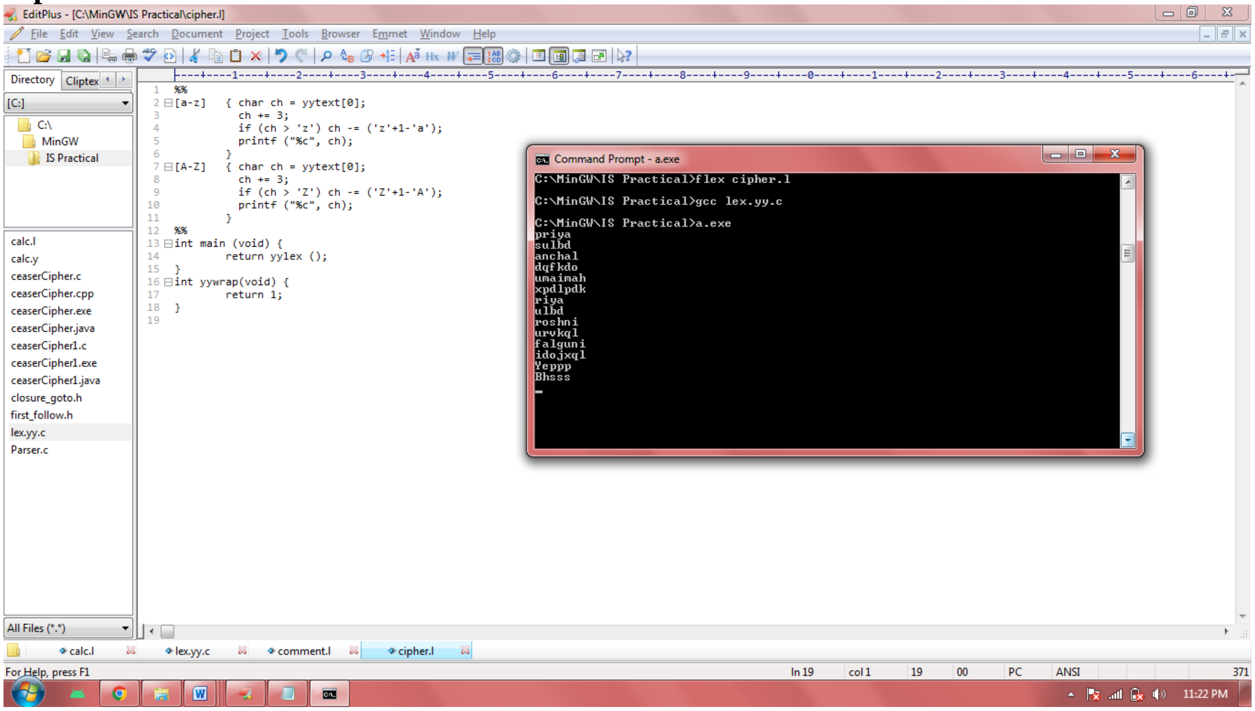
```
C:\MinGW\IS Practical>flex comment.l

C:\MinGW\IS Practical>gcc lex.yy.c

C:\MinGW\IS Practical>a.exe

0
C:\MinGW\IS Practical>
```

b. Ceasor Cypher

Code:

```
%%
[a-z]       { char ch = yytext[0];
          ch += 3;
          if (ch > 'z') ch -= ('z'+1-'a');
          printf ("%c", ch);
        }
[A-Z]       { char ch = yytext[0];
          ch += 3;
          if (ch > 'Z') ch -= ('Z'+1-'A');
          printf ("%c", ch);
        }
%%
int main (void) {
        return yylex ();
}
int yywrap(void) {
        return 1;
}
```

Output:



```
%%
[a-z]   { char ch = yytext[0];
          ch += 3;
          if (ch > 'z') ch -= ('z'+1-'a');
          printf ("%c", ch);
        }
[A-Z]   { char ch = yytext[0];
          ch += 3;
          if (ch > 'Z') ch -= ('Z'+1-'A');
          printf ("%c", ch);
        }
%%
int main (void) {
        return yylex ();
}
int yywrap(void) {
        return 1;
}
```

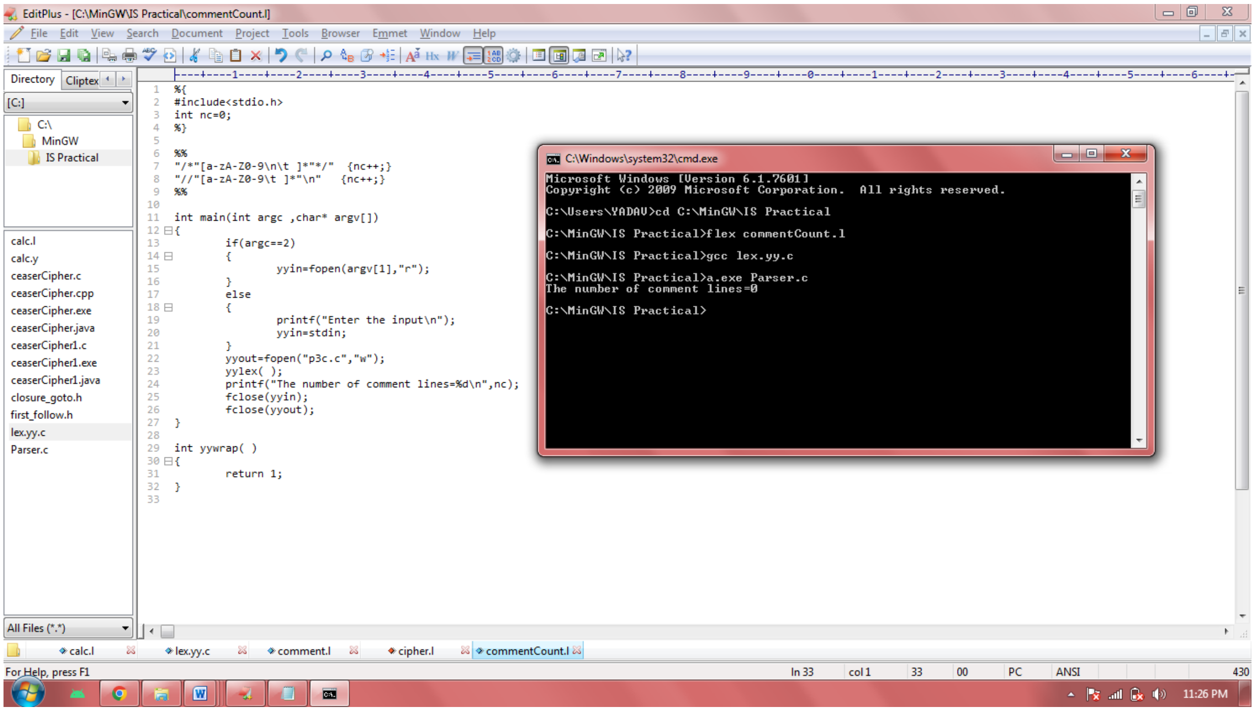
```
C:\MinGW\IS Practical>flex cipher.l

C:\MinGW\IS Practical>gcc lex.yy.c

C:\MinGW\IS Practical>a.exe
priya
sulbd
anchal
dqfkdo
umaimah
xpdlpdk
riya
ulbd
roshni
urvkql
falguni
idojxql
Yeppp
Bhsss
_
```

c. Extract single and multiline comments from C Program

CODE:

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

%%
"/*"[a-zA-Z0-9\n\t ]*"*/"  {nc++;}
"//"[a-zA-Z0-9\t ]*"\n"   {nc++;}
%%
int main(int argc ,char* argv[])
{
    if(argc==2){
              yyin=fopen(argv[1],"r");
    }
    else{
              printf("Enter the input\n");
              yyin=stdin;
    }
    yyout=fopen("p3c.c","w");
    yylex( );
    printf("The number of comment lines=%d\n",nc);
    fclose(yyin);
    fclose(yyout);
}
int yywrap( ){
    return 1;
}
```

OUTPUT:



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

%%
"/*"[a-zA-Z0-9\n\t ]*"*/"  {nc++;}
"//"[a-zA-Z0-9\t ]*"\n"   {nc++;}
%%

int main(int argc ,char* argv[])
{
        if(argc==2)
        {
                yyin=fopen(argv[1],"r");
        }
        else
        {
                printf("Enter the input\n");
                yyin=stdin;
        }
        yyout=fopen("p3c.c","w");
        yylex( );
        printf("The number of comment lines=%d\n",nc);
        fclose(yyin);
        fclose(yyout);
}

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

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\YADAV>cd C:\MinGW\IS Practical

C:\MinGW\IS Practical>flex commentCount.l

C:\MinGW\IS Practical>gcc lex.yy.c

C:\MinGW\IS Practical>a.exe Parser.c
The number of comment lines=0

C:\MinGW\IS Practical>
```

PRACTICAL 4

AIM: Implement following Programs Using Lex

a. Convert Roman to Decimal

Code:

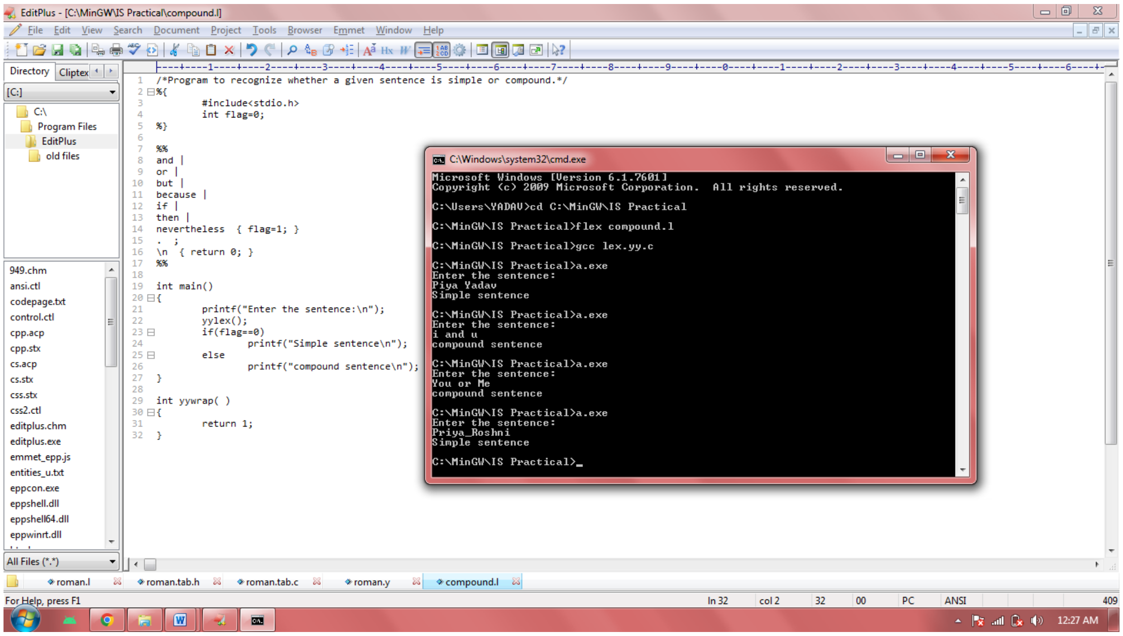
```
WS      [ \t]+

%%
        int total=0;

I       total += 1;
IV      total += 4;
V       total += 5;
IX      total += 9;
X       total += 10;
XL      total += 40;
L       total += 50;
XC      total += 90;
C       total += 100;
CD      total += 400;
D       total += 500;
CM      total += 900;
M       total += 1000;

{WS}    |
\n      return total;
%%
int main (void) {
  int first, second;

  first = yylex ();
  second = yylex ();

  printf ("%d + %d = %d\n", first, second, first+second);
  return 0;
 }
```

b. Check weather given statement is compound or simple

Code:

```
/*Program to recognize whether a given sentence is simple or compound.*/
%{
        #include<stdio.h>
        int flag=0;
%}

%%
and |
or |
but |
because |
if |
then |
nevertheless  { flag=1; }
.  ;
\n  { return 0; }
%%

int main(){
        printf("Enter the sentence:\n");
        yylex();
        if(flag==0)
                printf("Simple sentence\n");
        else
                printf("compound sentence\n");
}
int yywrap( ){
        return 1;
}
```

OUTPUT



```
/*Program to recognize whether a given sentence is simple or compound.*/
%{
        #include<stdio.h>
        int flag=0;
%}

%%
and |
or |
but |
because |
if |
then |
nevertheless  { flag=1; }
.  ;
\n  { return 0; }
%%

int main()
{
        printf("Enter the sentence:\n");
        yylex();
        if(flag==0)
                printf("Simple sentence\n");
        else
                printf("compound sentence\n");
}

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

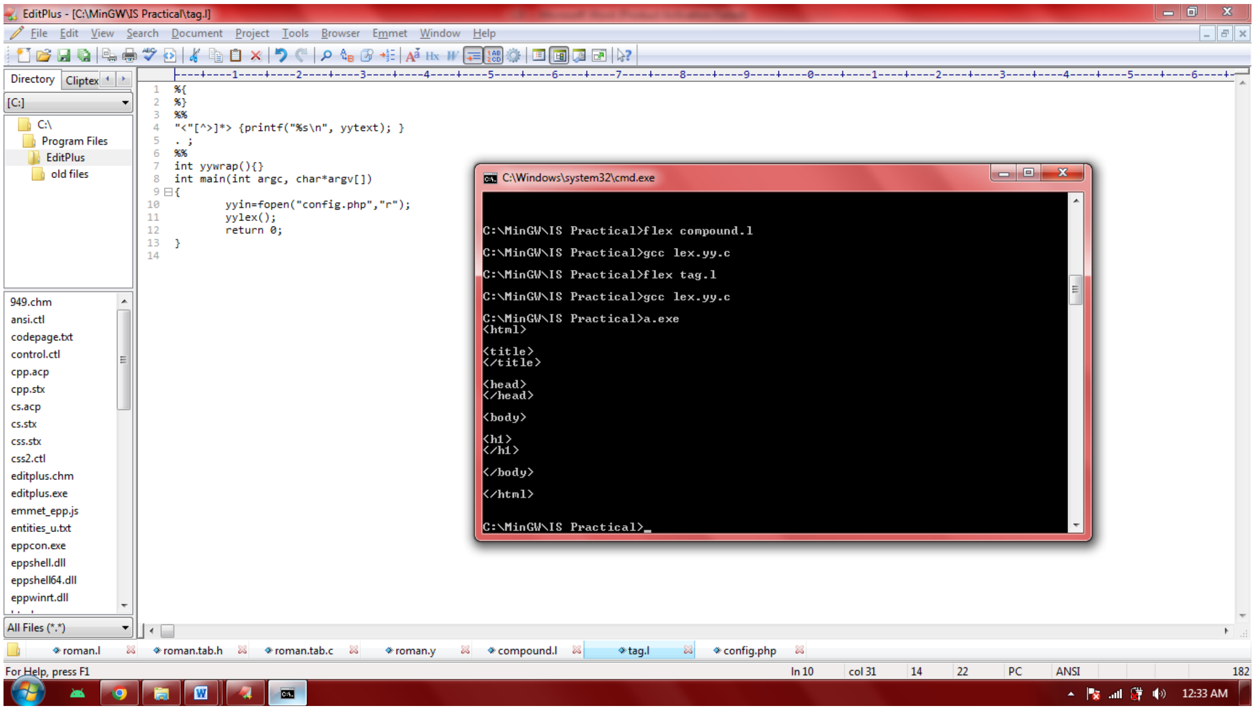
```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\YADAV>cd C:\MinGW\IS Practical

C:\MinGW\IS Practical>flex compound.l

C:\MinGW\IS Practical>gcc lex.yy.c

C:\MinGW\IS Practical>a.exe
Enter the sentence:
Piya Yadav
Simple sentence

C:\MinGW\IS Practical>a.exe
Enter the sentence:
i and u
compound sentence

C:\MinGW\IS Practical>a.exe
Enter the sentence:
You or Me
compound sentence

C:\MinGW\IS Practical>a.exe
Enter the sentence:
Priya_Roshni
Simple sentence

C:\MinGW\IS Practical>_
```

c. Extract html tags from .html file

CODE:

```
%{
%}
%%
"<"[^>]*> {printf("%s\n", yytext); }
. ;
%%
int yywrap(){}
int main(int argc, char*argv[])
{
        yyin=fopen("config.php","r");
        yylex();
        return 0;
}
```

OUTPUT:



```
%{
%}
%%
"<"[^>]*> {printf("%s\n", yytext); }
. ;
%%
int yywrap(){}
int main(int argc, char*argv[])
{
        yyin=fopen("config.php","r");
        yylex();
        return 0;
}
```

```
C:\MinGW\IS Practical>flex compound.l

C:\MinGW\IS Practical>gcc lex.yy.c

C:\MinGW\IS Practical>flex tag.l

C:\MinGW\IS Practical>gcc lex.yy.c

C:\MinGW\IS Practical>a.exe
<html>

<title>
</title>

<head>
</head>

<body>

<h1>
</h1>

</body>

</html>


C:\MinGW\IS Practical>_
```

PRACTICAL 5

AIM: Implementation of Recursive Descent Parser without backtracking

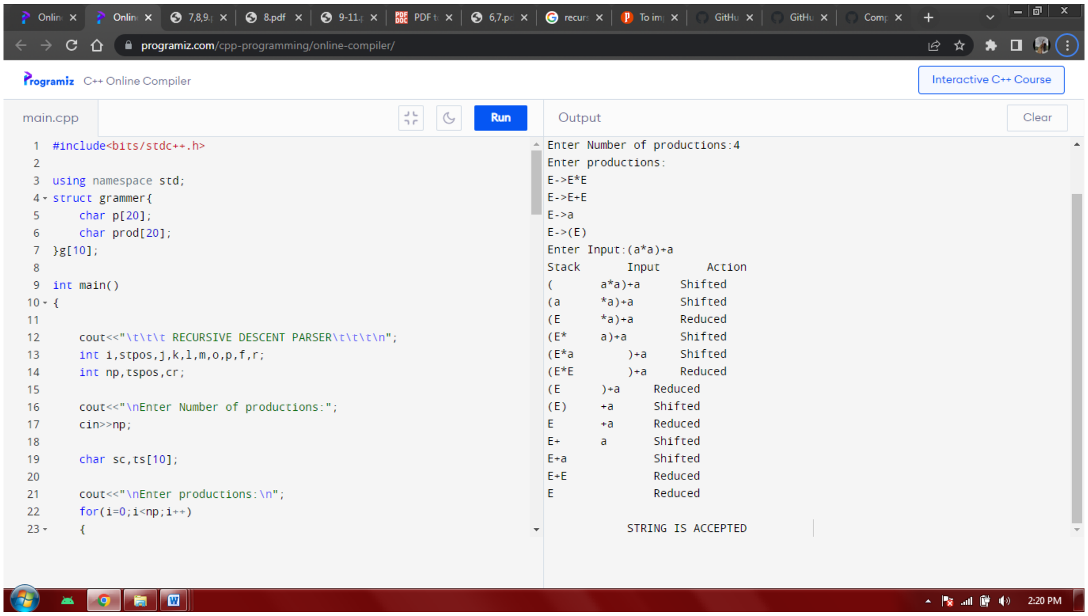
CODE:

```
#include<bits/stdc++.h>

using namespace std;
struct grammer{
   char p[20];
   char prod[20];
}g[10];

int main()
{

   cout<<"\t\t\t RECURSIVE DESCENT PARSER\t\t\t\n";
   int i,stpos,j,k,l,m,o,p,f,r;
   int np,tspos,cr;

   cout<<"\nEnter Number of productions:";
   cin>>np;

   char sc,ts[10];

   cout<<"\nEnter productions:\n";
   for(i=0;i<np;i++)
   {
      cin>>ts;
      strncpy(g[i].p,ts,1);
      strcpy(g[i].prod,&ts[3]);
   }

   char ip[10];

   cout<<"\nEnter Input:";
   cin>>ip;

   int lip=strlen(ip);

   char stack[10];

   stpos=0;
   i=0;

   //moving input
   sc=ip[i];
   stack[stpos]=sc;
   i++;stpos++;

   cout<<"\n\nStack\t\tInput\t\tAction";
```

```
do
{
   r=1;
   while(r!=0)
   {
      cout<<"\n";
      for(p=0;p<stpos;p++)
      {
         cout<<stack[p];
      }
      cout<<"\t\t";
      for(p=i;p<lip;p++)
      {
         cout<<ip[p];
      }

      if(r==2)
      {
         cout<<"\t\tReduced";
      }
      else
      {
         cout<<"\t\tShifted";
      }
      r=0;

      //try reducing

      for(k=0;k<stpos;k++)
      {
         f=0;

         for(l=0;l<10;l++)
         {
            ts[l]='\0';
         }

         tspos=0;
         for(l=k;l<stpos;l++) //removing first caharcter
         {
            ts[tspos]=stack[l];
            tspos++;
         }

         //now compare each possibility with production
         for(m=0;m<np;m++)
         {
            cr = strcmp(ts,g[m].prod);

            //if cr is zero then match is found
            if(cr==0)
            {
```

```
                        for(l=k;l<10;l++) //removing matched part from stack
                        {
                            stack[l]='\0';
                            stpos--;
                        }

                        stpos=k;

                        //concatinate the string
                        strcat(stack,g[m].p);
                        stpos++;
                        r=2;
                    }
                }
            }
        }
        //moving input
        sc=ip[i];
        stack[stpos]=sc;
        i++;stpos++;

    }while(strlen(stack)!=1 && stpos!=lip);

    if(strlen(stack)==1)
    {
        cout<<"\n\n \t\t\tSTRING IS ACCEPTED\t\t\t";
    }
    else
            cout<<"\n\n \t\t\tSTRING IS REJECTED\t\t\t";
    return 0;
}
```

OUTPUT:



```
#include<bits/stdc++.h>

using namespace std;
struct grammer{
    char p[20];
    char prod[20];
}g[10];

int main()
{

    cout<<"\t\t\t RECURSIVE DESCENT PARSER\t\t\t\n";
    int i,stpos,j,k,l,m,o,p,f,r;
    int np,tspos,cr;

    cout<<"\nEnter Number of productions:";
    cin>>np;

    char sc,ts[10];

    cout<<"\nEnter productions:\n";
    for(i=0;i<np;i++)
    {
```

```
Enter Number of productions:4
Enter productions:
E->E*E
E->E+E
E->a
E->(E)
Enter Input:(a*a)+a
Stack       Input       Action
(       a*a)+a      Shifted
(a      *a)+a       Shifted
(E      *a)+a       Reduced
(E*     a)+a        Shifted
(E*a        )+a     Shifted
(E*E        )+a     Reduced
(E      )+a     Reduced
(E)     +a      Shifted
E       +a      Reduced
E+      a       Shifted
E+a             Shifted
E+E             Reduced
E               Reduced

            STRING IS ACCEPTED
```

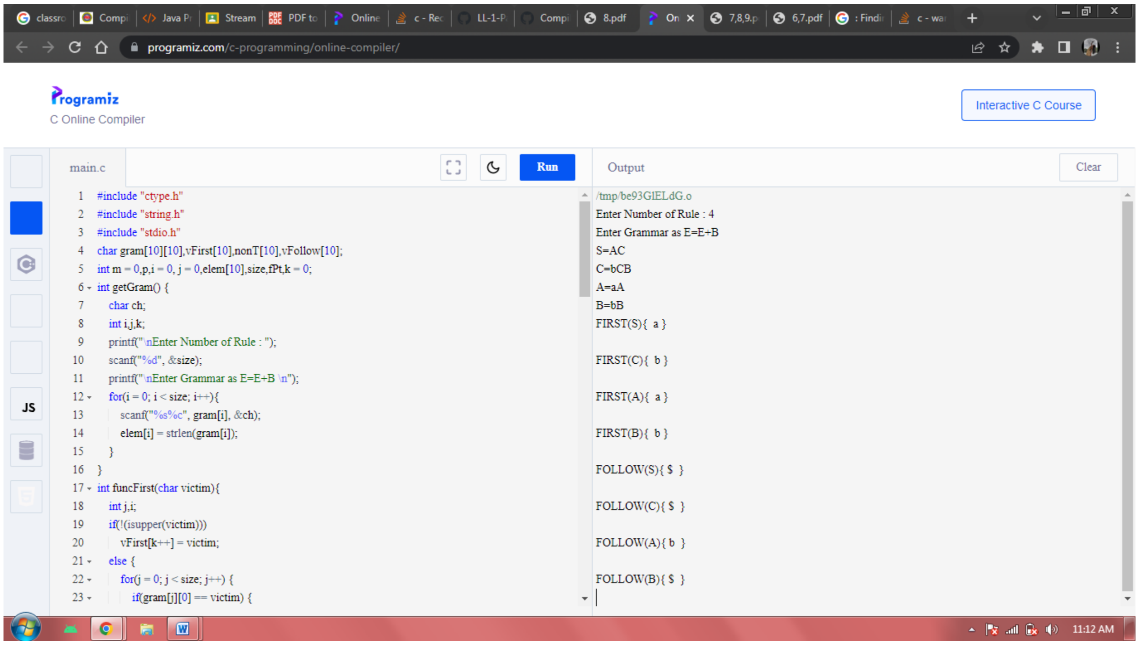
PRACTICAL 6

AIM: Finding “First” set

CODE:

```
#include "ctype.h"
#include "string.h"
#include "stdio.h"
char gram[10][10],vFirst[10],nonT[10],vFollow[10];
int m = 0,p,i = 0, j = 0,elem[10],size,fPt,k = 0;
int getGram() {
    char ch;
    int i,j,k;
    printf("\nEnter Number of Rule : ");
    scanf("%d", &size);
    printf("\nEnter Grammar as E=E+B \n");
    for(i = 0; i < size; i++){
        scanf("%s%c", gram[i], &ch);
        elem[i] = strlen(gram[i]);
    }
}
int funcFirst(char victim){
    int j,i;
    if(!(isupper(victim)))
        vFirst[k++] = victim;
    else {
        for(j = 0; j < size; j++) {
            if(gram[j][0] == victim) {
                if(gram[j][2] == '$')
                    vFirst[k++] = '$';
                else if(islower(gram[j][2]))
                    vFirst[k++] = gram[j][2];
                else
                    funcFirst(gram[j][2]);
            }
        }
    }
}
void funFollow(char);
void first(char victim) {
    int k;
    if(!(isupper(victim)))
    vFollow[m++] = victim;
    for(k = 0; k < size; k++) {
        if(gram[k][0] == victim) {
            if(gram[k][2] == '$')
                funFollow(gram[i][0]);
            else if(islower(gram[k][2]))
                vFollow[m++] = gram[k][2];
            else
            first(gram[k][2]);
        }
    }
}
void funFollow(char victim) {
    if(gram[0][0] == victim)
    vFollow[m++] = '$';
    for(i = 0; i < size; i++) {
        for(j = 2; j < strlen(gram[i]); j++) {
            if(gram[i][j] == victim) {
                if(gram[i][j+1] != '\0')
                first(gram[i][j+1]);if(gram[i][j+1] == '\0' && victim != gram[i][0])
                funFollow(gram[i][0]);
            }
        }
    }
}
int main() {
    int i,j,l=0;
    getGram();
    for(i = 0; i < size; i++) {
        for (j = 0; j < i; j++) {
            if (gram[i][0] == gram[j][0])
                break;
            }
        if (i == j) {
            nonT[l] = gram[i][0];
            l++;
        }
    }
    for(i = 0; i < l; i++) {
        k = 0;
        funcFirst(nonT[i]);
        printf("\nFIRST(%c){ ", nonT[i]);
        for(j = 0; j < strlen(vFirst); j++)
        printf(" %c", vFirst[j]);
        printf(" }\n");
    }
    int s = 0;
    for(s = 0; s < l; s++) {
        m = 0;
        printf("\nFOLLOW(%c){ ", nonT[s]);
        funFollow(nonT[s]);
        for(i = 0; i < m; i++)
        printf("%c ", vFollow[i]);
        printf(" }\n");
    }
}
```

OUTPUT:



```c
#include "ctype.h"
#include "string.h"
#include "stdio.h"
char gram[10][10],vFirst[10],nonT[10],vFollow[10];
int m = 0,p,i = 0, j = 0,elem[10],size,fPt,k = 0;
int getGram() {
    char ch;
    int i,j,k;
    printf("\nEnter Number of Rule : ");
    scanf("%d", &size);
    printf("\nEnter Grammar as E=E+B \n");
    for(i = 0; i < size; i++){
        scanf("%s%c", gram[i], &ch);
        elem[i] = strlen(gram[i]);
    }
}
int funcFirst(char victim){
    int j,i;
    if(!(isupper(victim)))
        vFirst[k++] = victim;
    else {
        for(j = 0; j < size; j++) {
            if(gram[j][0] == victim) {
```

```
/tmp/be93GlELdG.o
Enter Number of Rule : 4
Enter Grammar as E=E+B
S=AC
C=bCB
A=aA
B=bB
FIRST(S){ a }

FIRST(C){ b }

FIRST(A){ a }

FIRST(B){ b }

FOLLOW(S){ $ }

FOLLOW(C){ $ }

FOLLOW(A){ b }

FOLLOW(B){ $ }
```

PRACTICAL 7

AIM: Generate 3-tuple intermediate code for given infix expression

- **Three address code** is a type of intermediate code which is easy to generate and can be easily converted to machine code.
- It makes use of at most three addresses and one operator to represent an expression and the value computed at each instruction is stored in temporary variable generated by compiler.
- The compiler decides the order of operation given by three address code.
- **General representation –**
  a = b op c

  Where a, b or c represents operands like names, constants or compiler generated temporaries and op represents the operator

- **Implementation of Three Address Code –**
  There are 3 representations of three address code namely
  1. Quadruple
  2. Triples
  3. Indirect Triples

1. Quadruple –
It is structure with consist of 4 fields namely op, arg1, arg2 and result. op denotes the operator and arg1 and arg2 denotes the two operands and result is used to store the result of the expression.

2. Triples –
This representation doesn't make use of extra temporary variable to represent a single operation instead when a reference to another triple's value is needed, a pointer to that triple is used. So, it consist of only three fields namely op, arg1 and arg2.

3. Indirect Triples –
This representation makes use of pointer to the listing of all references to computations which is made separately and stored. Its similar in utility as compared to quadruple representation but requires less space than it. Temporaries are implicit and easier to rearrange code.

CODE:

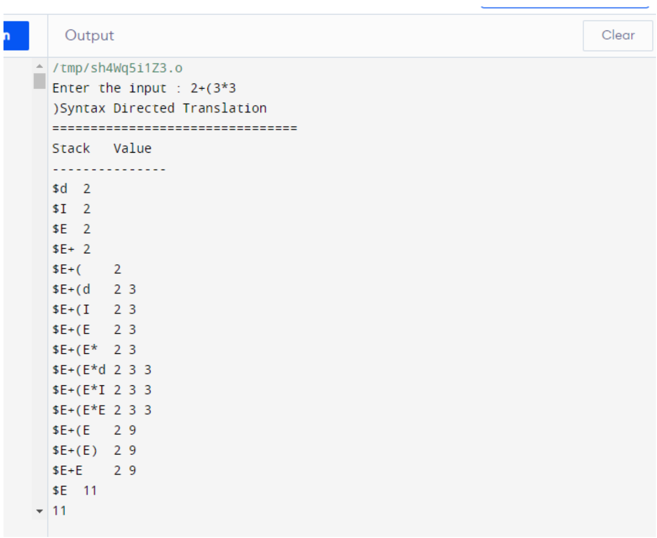
```
#include<bits/stdc++.h>
#include<string>
using namespace std;
char stac[20],val1[20],sym[20];
int val[20];
int top1=-1,top2=-1,top3=-1;
string input;
void print_stac(){
for(int i=0;i<=top1;i++){
cout<<stac[i];
}
}
void print_val(){
for(int i=0;i<=top2;i++){
cout<<val[i]<<" ";
```

```
}
}
void print_val1(){
for(int i=0;i<=top2;i++){
cout<<val1[i];
}}
void sdt(){
stac[0] = '$';top1=0;
input[input.length()]='$';
cout<<"Stack\tValue\n---------------\n";
for(int i=0;i<input.length();i++){
if(input[i]>='0' && input[i]<='9'){
stac[top1+1] = 'd';
top1+=1;
val[top2+1] = int(input[i])-48;
top2+=1;
print_stac();
cout<<"\t";
print_val();
cout<<endl;
}
else{
stac[top1+1] = input[i];
top1+=1;
print_stac();
cout<<"\t";
print_val();
cout<<endl;
}
if(stac[top1]=='d' && stac[top1-1]=='I'){
stac[top1-1] = 'I';
top1-=1;
val[top2-1] = 10*val[top2-1] + val[top2];
top2-=1;
print_stac();
cout<<"\t";
print_val();
cout<<endl;
}
if(stac[top1]=='d'){
stac[top1]='I';
print_stac();
cout<<"\t";
print_val();
cout<<endl;
}
if(stac[top1]=='I' && (input[i+1]<'0' || input[i+1]>'9')){ stac[top1] = 'E';
print_stac();
cout<<"\t";
print_val();
cout<<endl;
}
```

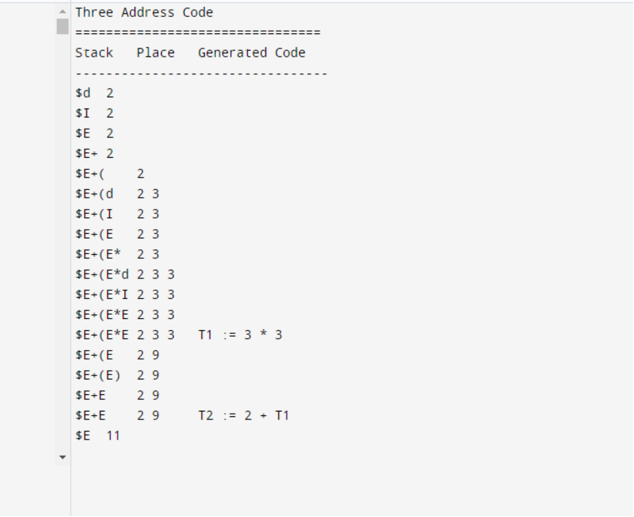
```
if(stac[top1]==')' && stac[top1-1]=='E' && stac[top1-2]=='('){ stac[top1-2]='E';
top1-=2;
print_stac();
cout<<"\t";
print_val();
cout<<endl;
}
if(stac[top1]=='E' && stac[top1-1]=='+' && stac[top1-2]=='E'){ stac[top1-2]='E';
top1-=2;
val[top2-1] = val[top2]+val[top2-1];
top2-=1;
print_stac();
cout<<"\t";
print_val();
cout<<endl;
}
if(stac[top1]=='E' && stac[top1-1]=='*' && stac[top1-2]=='E'){ stac[top1-2]='E';
top1-=2;
val[top2-1] = val[top2]*val[top2-1];
top2-=1;
print_stac();
cout<<"\t";
print_val();
cout<<endl;
}
if(stac[top1]=='$' && stac[top1-1]=='E'){
stac[top1-1]='S';
top1-=1;
print_stac();
cout<<"\t";
print_val();
cout<<endl;
}
cout<<val[top2]<<endl;
}
void convert(){
stac[0] = '$';top1=0;top2=-1;
input[input.length()]='$';
cout<<"Stack\tPost-fix\n---------------\n";
for(int i=0;i<input.length();i++){
if(input[i]>='0' && input[i]<='9'){
stac[top1+1] = 'd';
top1+=1;
val1[top2+1] = input[i];
top2+=1;
print_stac();
cout<<"\t";
print_val1();
cout<<endl;
}
else{
stac[top1+1] = input[i];
```

```
if(input[i]=='*' || input[i]=='+'){
sym[top3+1]=input[i];
op3+=1;
}
p1+=1;
rint_stac();
cout<<"\t";
rint_val1();
cout<<endl;
f(stac[top1]=='d' && stac[top1-1]=='I'){

ac[top1-1] = 'I';
top1-=1;
print_stac();
cout<<"\t";
print_val1();
cout<<endl;
if(stac[top1]=='d'){
stac[top1]='I';
print_stac();
out<<"\t";
print_val1();
cout<<endl;
}f(stac[top1]=='I' && (input[i+1]<'0' || input[i+1]>'9')){ stac[top1] = 'E';
rnt_stac();
cout<<"\t";
print_val1();
out<<endl;
}
if(stac[top1]==')' && stac[top1-1]=='E' && stac[top1-2]=='('){ stac[top1-2]='E';
top1-=2;
val1[top2+1]=sym[top3];
op3-=1;top2+=1;
print_stac();
cout<<"\t";
int_val1();
out<<endl;

(stac[top1]=='E' && stac[top1-1]=='+' && stac[top1-2]=='E'){ stac[top1-2]='E';
top1-=2;
int_stac();
ut<<"\t";
print_val1();
out<<endl;
(stac[top1]=='E' && stac[top1-1]=='*' && stac[top1-2]=='E'){ stac[top1-2]='E';
top1-=2;
//          val1[top2+1] = '*';
//          top2+=1;
rint_stac();
cout<<"\t";
rint_val1();
cout<<endl;
}
f(stac[top1]=='$' && stac[top1-1]=='E'){

ac[top1-1]='S';
top1-=1;
print_stac();
out<<"\t";
print_val1();
cout<<endl;
}
print_stac();
out<<"\t";
print_val1();
hile(top3!=-1){
cout<<sym[top3];
top3-=1;
}
cout<<endl;
void threeAddressCode(){
stac[0] = '$';top1=0;top2=-1;
nt x=1;
vector<vector<string> > v;
input[input.length()]='$';
out<<"Stack\tPlace\tGenerated Code\n----------------------------------\n";
for(int i=0;i<input.length();i++){
}
}
}
int main(){
//get_gram();
cout<<"Enter the input : ";
cin>>input;
cout<<"Syntax Directed Translation\n=================================\n"; sdt();
cout<<"Infix to postfix\n=================================\n"; convert();
cout<<"Three Address Code\n=================================\n";
threeAddressCode();
return 0;
}
```

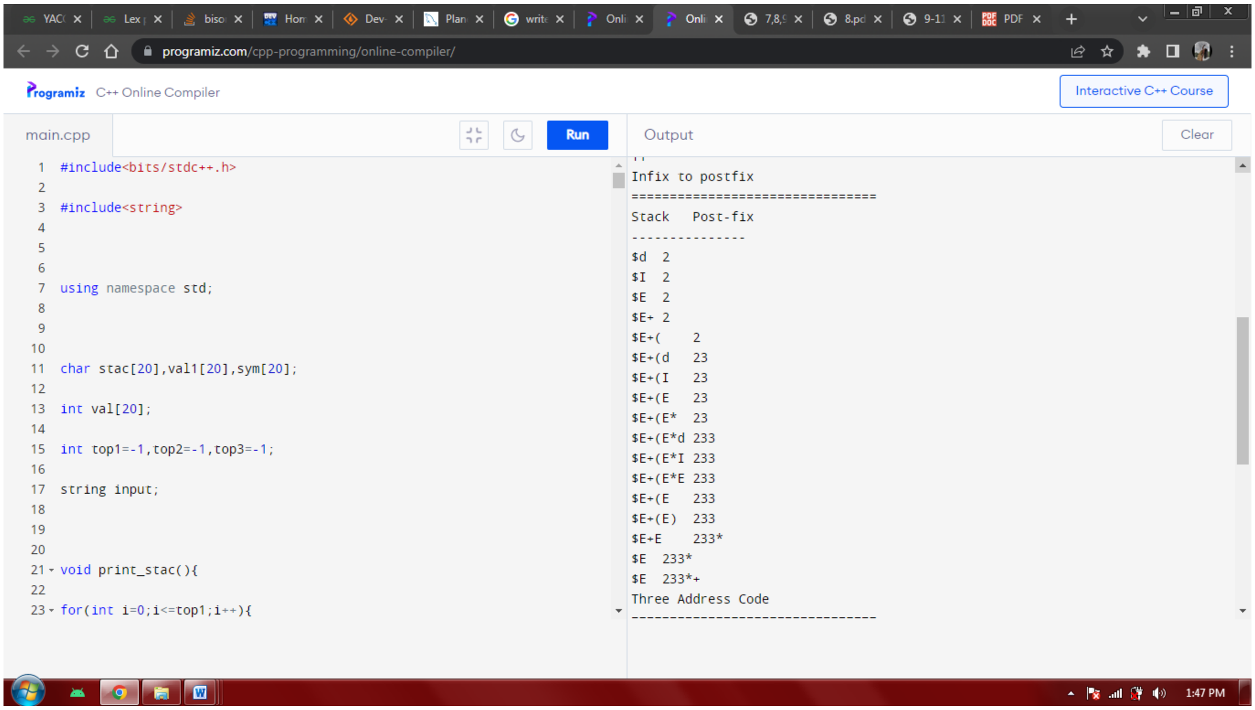
OUTPUT:



```
/tmp/sh4Wq5i1Z3.o
Enter the input : 2+(3*3
)Syntax Directed Translation
===============================
Stack   Value
---------------
$d  2
$I  2
$E  2
$E+ 2
$E+(    2
$E+(d   2 3
$E+(I   2 3
$E+(E   2 3
$E+(E*  2 3
$E+(E*d 2 3 3
$E+(E*I 2 3 3
$E+(E*E 2 3 3
$E+(E   2 9
$E+(E)  2 9
$E+E    2 9
$E  11
11
```



```
Three Address Code
================================
Stack   Place   Generated Code
---------------------------------
$d  2
$I  2
$E  2
$E+ 2
$E+(    2
$E+(d   2 3
$E+(I   2 3
$E+(E   2 3
$E+(E*  2 3
$E+(E*d 2 3 3
$E+(E*I 2 3 3
$E+(E*E 2 3 3
$E+(E*E 2 3 3   T1  := 3 * 3
$E+(E   2 9
$E+(E)  2 9
$E+E    2 9
$E+E    2 9     T2  := 2 + T1
$E  11
```



```
#include<bits/stdc++.h>

#include<string>



using namespace std;



char stac[20],val1[20],sym[20];

int val[20];

int top1=-1,top2=-1,top3=-1;

string input;



void print_stac(){

for(int i=0;i<=top1;i++){
```

```
Infix to postfix
================================
Stack   Post-fix
---------------
$d  2
$I  2
$E  2
$E+ 2
$E+(    2
$E+(d   23
$E+(I   23
$E+(E   23
$E+(E*  23
$E+(E*d 233
$E+(E*I 233
$E+(E*E 233
$E+(E   233
$E+(E)  233
$E+E    233*
$E  233*
$E  233*+
Three Address Code
--------------------------------
```

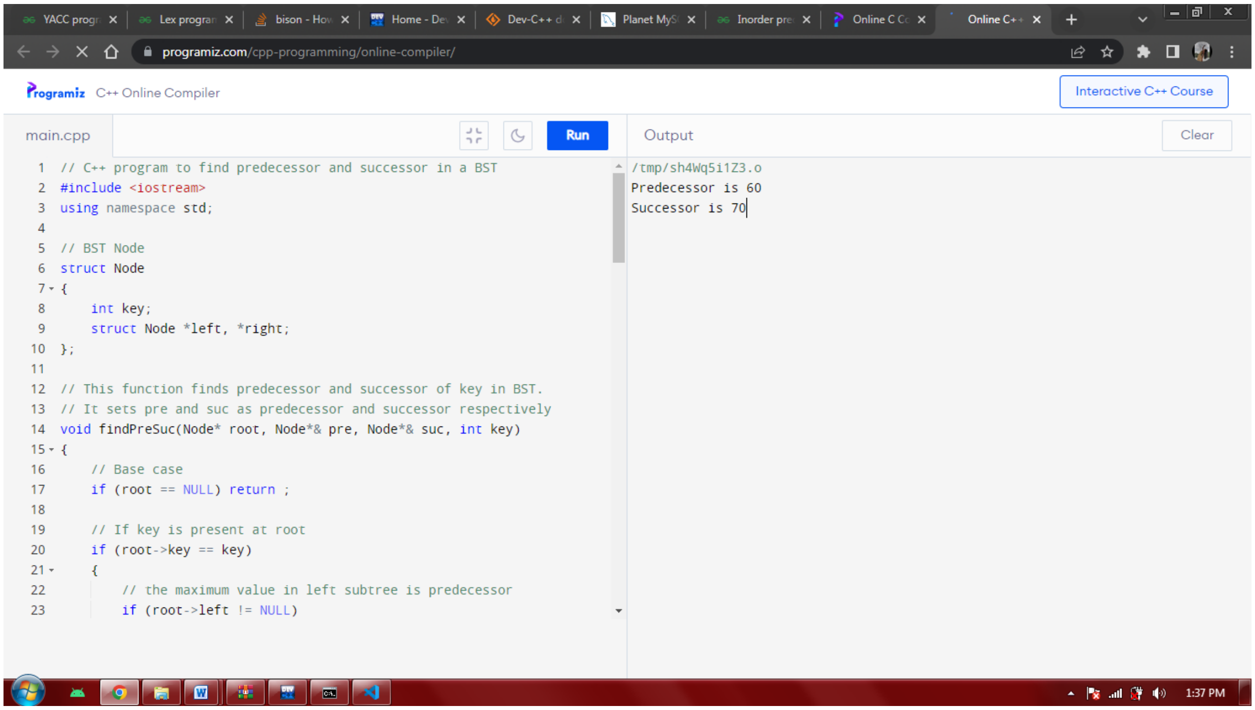
PRACTICAL 8

AIM: Extract Predecessor and Successor from given Control Flow Graph

CODE:

```
// program to find predecessor and successor in a BST
#include <iostream>
using namespace std;
struct Node
{
        int key;
        struct Node *left, *right;
};
void findPreSuc(Node* root, Node*& pre, Node*& suc, int key)
{
        // Base case
        if (root == NULL) return ;

        // If key is present at root
        if (root->key == key)
        {
                // the maximum value in left subtree is predecessor
                if (root->left != NULL)
                {
                        Node* tmp = root->left;
                        while (tmp->right)
                                tmp = tmp->right;
                        pre = tmp ;
                }

                // the minimum value in right subtree is successor
                if (root->right != NULL)
                {
                        Node* tmp = root->right ;
                        while (tmp->left)
                                tmp = tmp->left ;
                        suc = tmp ;
                }
                return ;
        }

        // If key is smaller than root's key, go to left subtree
        if (root->key > key)
        {
                suc = root ;
                findPreSuc(root->left, pre, suc, key) ;
        }
        else // go to right subtree
        {
                pre = root ;
                findPreSuc(root->right, pre, suc, key) ;
        }
}
Node *newNode(int item)
{
        Node *temp = new Node;
        temp->key = item;
        temp->left = temp->right = NULL;
        return temp;
}

/* A utility function to insert a new node with given key in BST */
Node* insert(Node* node, int key)
{
        if (node == NULL) return newNode(key);
        if (key < node->key)
                node->left = insert(node->left, key);
        else
                node->right = insert(node->right, key);
        return node;
}
// Driver program to test above function
int main()
{
        int key = 65; //Key to be searched in BST
        Node *root = NULL;
        root = insert(root, 50);
        insert(root, 30);
        insert(root, 20);
        insert(root, 40);
        insert(root, 70);
        insert(root, 60);
        insert(root, 80);
        Node* pre = NULL, *suc = NULL;
        findPreSuc(root, pre, suc, key);
        if (pre != NULL)
        cout << "Predecessor is " << pre->key << endl;
        else
        cout << "No Predecessor";
        if (suc != NULL)
        cout << "Successor is " << suc->key;
        else
        cout << "No Successor";
        return 0;
}
```

OUTPUT:



```
// C++ program to find predecessor and successor in a BST
#include <iostream>
using namespace std;

// BST Node
struct Node
{
    int key;
    struct Node *left, *right;
};

// This function finds predecessor and successor of key in BST.
// It sets pre and suc as predecessor and successor respectively
void findPreSuc(Node* root, Node*& pre, Node*& suc, int key)
{
    // Base case
    if (root == NULL) return ;

    // If key is present at root
    if (root->key == key)
    {
        // the maximum value in left subtree is predecessor
        if (root->left != NULL)
```

```
/tmp/sh4Wq5i1Z3.o
Predecessor is 60
Successor is 70
```

PRACTICAL 9

AIM: Introduction to YACC and generate Calculator Program

YACC

- YACC stands for **Yet Another Compiler Compiler**.
- YACC provides a tool to produce a parser for a given grammar.
- YACC is a program designed to compile a LALR (1) grammar.
- It is used to produce the source code of the syntactic analyzer of the language produced by LALR (1) grammar.
- The input of YACC is the rule or grammar and the output is a C program.

These are some points about YACC:

Input: A CFG- file.y

Output: A parser y.tab.c (yacc)

- The output file "file.output" contains the parsing tables.
- The file "file.tab.h" contains declarations.
- The parser called the yyparse ().
- Parser expects to use a function called yylex () to get tokens.

CODE:

File Name: Calc.l

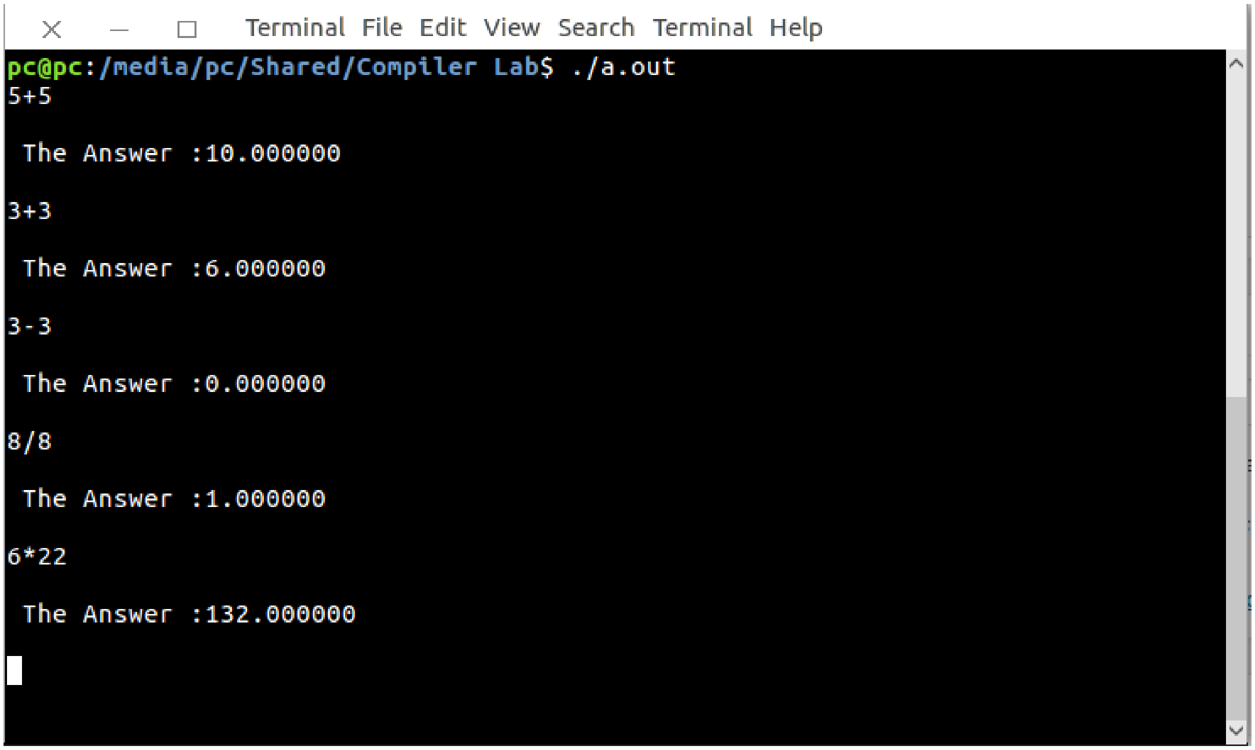
```
%{
/* Definition section */
#include<stdio.h>
#include "y.tab.h"
extern int yylval;
%}

/* Rule Section */
%%
[0-9]+ {
                    yylval=atoi(yytext);
                    return NUMBER;

          }
[\t] ;

[\n] return 0;

. return yytext[0];

%%

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

File Name: Calc.y

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

%token NUMBER

%left '+' '-'

%left '*' '/' '%'

%left '(' ')'

/* Rule Section */
%%

ArithmeticExpression: E{

                printf("\nResult=%d\n", $$);
                return 0;
                };
E:E'+'E {$$=$1+$3;}

|E'-'E {$$=$1-$3;}

|E'*'E {$$=$1*$3;}

|E'/'E {$$=$1/$3;}

|E'%'E {$$=$1%$3;}

|'('E')' {$$=$2;}

| NUMBER {$$=$1;}

;

%%

//driver code
void main()
{
printf("\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:



```
pc@pc:/media/pc/Shared/Compiler Lab$ ./a.out
5+5

 The Answer :10.000000

3+3

 The Answer :6.000000

3-3

 The Answer :0.000000

8/8

 The Answer :1.000000

6*22

 The Answer :132.000000

```

PRACTICAL 10

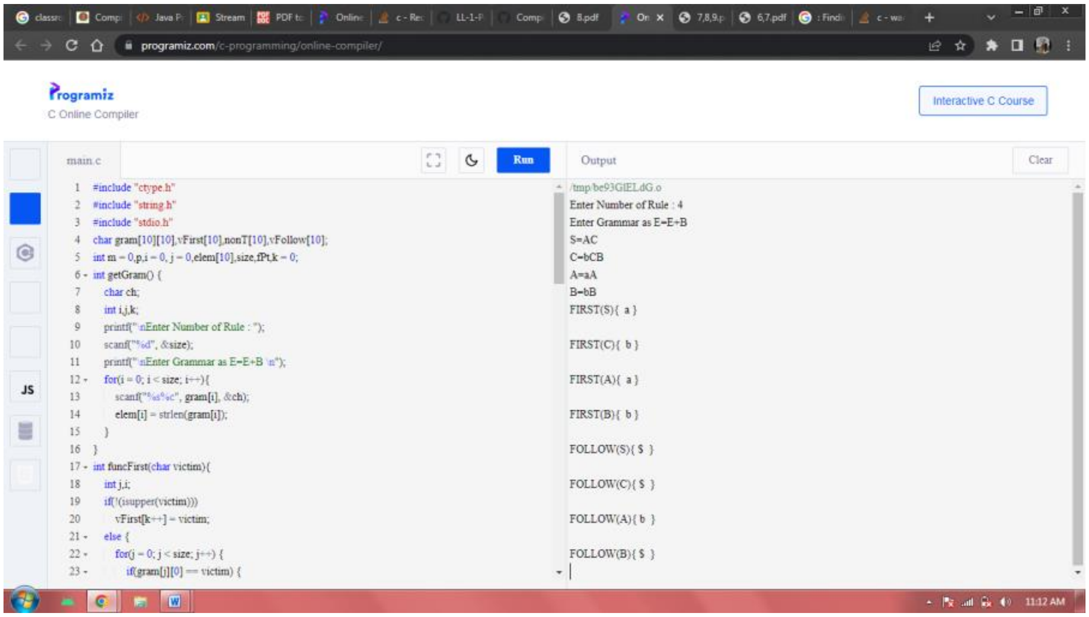
AIM: Finding “Follow” set

CODE:

```
#include "ctype.h"
#include "string.h"
#include "stdio.h"
char gram[10][10],vFirst[10],nonT[10],vFollow[10];
int m = 0,p,i = 0, j = 0,elem[10],size,fPt,k = 0;
int getGram() {
    char ch;
    int i,j,k;
    printf("\nEnter Number of Rule : ");
    scanf("%d", &size);
    printf("\nEnter Grammar as E=E+B \n");
    for(i = 0; i < size; i++){
        scanf("%s%c", gram[i], &ch);
        elem[i] = strlen(gram[i]);
    }}
int funcFirst(char victim){
    int j,i;
    if(!(isupper(victim)))
        vFirst[k++] = victim;
    else {
        for(j = 0; j < size; j++) {
            if(gram[j][0] == victim) {
                if(gram[j][2] == '$')
                    vFirst[k++] = '$';
                else if(islower(gram[j][2]))
                    vFirst[k++] = gram[j][2];
                else
                    funcFirst(gram[j][2]);
            } }
    }}
void funFollow(char);
void first(char victim) {
    int k;
    if(!(isupper(victim)))
    vFollow[m++] = victim;
    for(k = 0; k < size; k++) {
        if(gram[k][0] == victim) {
            if(gram[k][2] == '$')
                funFollow(gram[i][0]);
            else if(islower(gram[k][2]))
                vFollow[m++] = gram[k][2];
            else
            first(gram[k][2]);
        }
    }}
```

```
void funFollow(char victim) {
   if(gram[0][0] == victim)
   vFollow[m++] = '$';
   for(i = 0; i < size; i++) {
      for(j = 2; j < strlen(gram[i]); j++) {
         if(gram[i][j] == victim) {
            if(gram[i][j+1] != '\0')
            first(gram[i][j+1]);if(gram[i][j+1] == '\0' && victim != gram[i][0])
            funFollow(gram[i][0]);
         }
      }
   }
}
int main() {
   int i,j,l=0;
   getGram();
   for(i = 0; i < size; i++) {
      for (j = 0; j < i; j++) {
         if (gram[i][0] == gram[j][0])
            break;
         }
      if (i == j) {
         nonT[l] = gram[i][0];
         l++;
      }
   }
   for(i = 0; i < l; i++) {
      k = 0;
      funcFirst(nonT[i]);
      printf("\nFIRST(%c){ ", nonT[i]);
      for(j = 0; j < strlen(vFirst); j++)
      printf(" %c", vFirst[j]);
      printf(" }\n");
   }
   int s = 0;
   for(s = 0; s < l; s++) {
      m = 0;
      printf("\nFOLLOW(%c){ ", nonT[s]);
      funFollow(nonT[s]);
      for(i = 0; i < m; i++)
      printf("%c ", vFollow[i]);
      printf(" }\n");
   }
}
```

OUTPUT:



```c
#include "ctype.h"
#include "string.h"
#include "stdio.h"
char gram[10][10],vFirst[10],nonT[10],vFollow[10];
int m = 0,p,i = 0, j = 0,elem[10],size,fPt,k = 0;
int getGram() {
   char ch;
   int i,j,k;
   printf("\nEnter Number of Rule : ");
   scanf("%d", &size);
   printf("\nEnter Grammar as E=E+B \n");
   for(i = 0; i < size; i++){
      scanf("%s%c", gram[i], &ch);
      elem[i] = strlen(gram[i]);
   }
}
int funcFirst(char victim){
   int j,i;
   if(!(isupper(victim)))
      vFirst[k++] = victim;
   else {
      for(j = 0; j < size; j++) {
         if(gram[j][0] == victim) {
```

```
/tmp/be93GIELdG.o
Enter Number of Rule : 4
Enter Grammar as E=E+B
S=AC
C=bCB
A=aA
B=bB
FIRST(S){ a }

FIRST(C){ b }

FIRST(A){ a }

FIRST(B){ b }

FOLLOW(S){ $ }

FOLLOW(C){ $ }

FOLLOW(A){ b }

FOLLOW(B){ $ }
```

PRACTICAL 11

AIM: Implement a C program to implement LALR parsing.

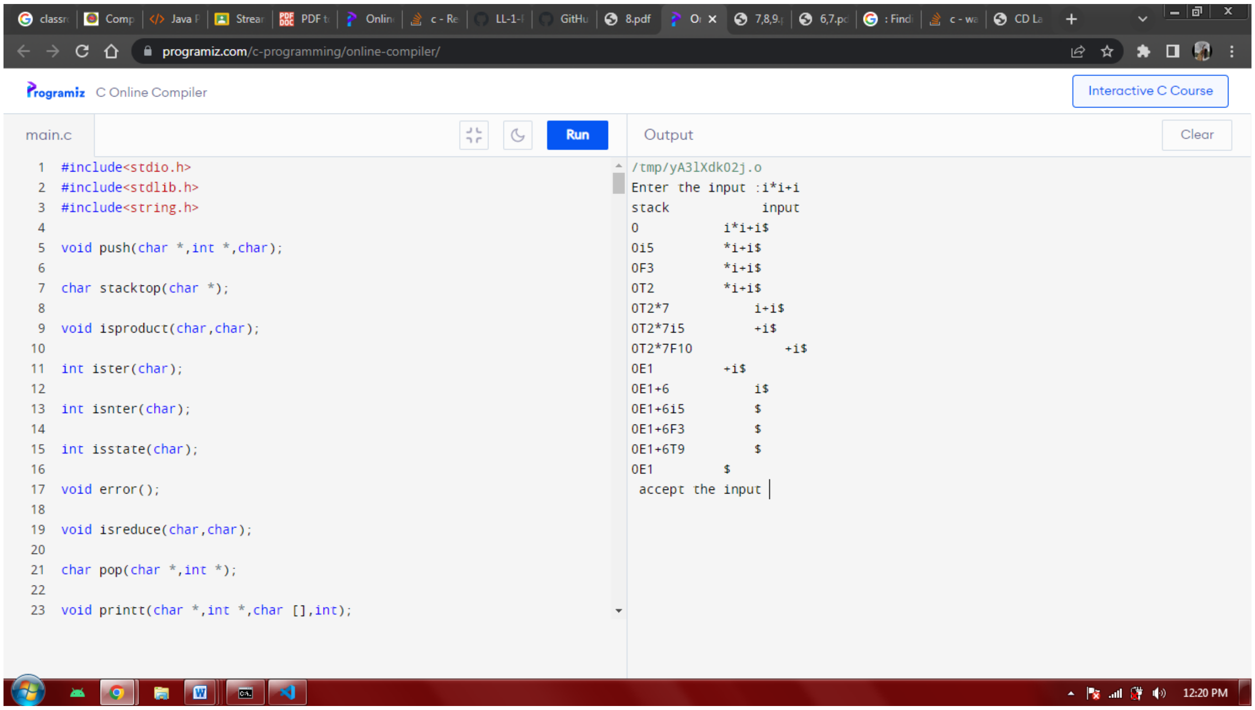
CODE:

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

void push(char *,int *,char);
char stacktop(char *);
void isproduct(char,char);
int ister(char);
int isnter(char);
int isstate(char);
void error();
void isreduce(char,char);
char pop(char *,int *);
void printt(char *,int *,char [],int);
void rep(char [],int);
struct action
{
char row[6][5];
};
const struct action A[12]={
{"sf","emp","emp","se","emp","emp"},
{"emp","sg","emp","emp","emp","acc"},
{"emp","rc","sh","emp","rc","rc"},
{"emp","re","re","emp","re","re"},
{"sf","emp","emp","se","emp","emp"},
{"emp","rg","rg","emp","rg","rg"},
{"sf","emp","emp","se","emp","emp"},
{"sf","emp","emp","se","emp","emp"},
{"emp","sg","emp","emp","sl","emp"},
"emp","rb","sh","emp","rb","rb"},
{"emp","rb","rd","emp","rd","rd"},
{"emp","rf","rf","emp","rf","rf"}
};

ruct gotol
{
char r[3][4];
};
const struct gotol G[12]={
"b","c","d"},
{"emp","emp","emp"},
{"emp","emp","emp"},
{"emp","emp","emp"},
{"i","c","d"},

"emp","emp","emp"},
"emp","j","d"},
{"emp","emp","k"},
{"emp","emp","emp"},
{"emp","emp","emp"},
};
char ter[6]={'i','+','*',')','(','$'};
char nter[3]={'E','T','F'};
char states[12]={'a','b','c','d','e','f','g','h','m','j','k','l'};
char stack[100];
int top=-1;
char temp[10];

ruct grammar

char left;
char right[5];
};
const struct grammar rl[6]={
{'E',"e+T"},
{'E',"T"},
{'T',"T*F"},
{'T',"F"},
{'F',"(E)"},
{'F',"i"},
};
void main()
{
char inp[80],x,p,dl[80],y,bl='a';
int i=0,j,k,l,n,m,c,len;

rintf(" Enter the input :");
scanf("%s",inp);
en=strlen(inp);
np[len]='$';
inp[len+1]='\0';
push(stack,&top,bl);
printf("\n stack \t\t\t input");
printt(stack,&top,inp,i);
do
x=inp[i];
p=stacktop(stack);
isproduct(x,p);
if(strcmp(temp,"emp")==0)
error();
if(strcmp(temp,"acc")==0)
break;
else
{
if(temp[0]=='s')
{
push(stack,&top,inp[i]);
push(stack,&top,temp[1]);
```

```
i++;
}
else
{
if(temp[0]=='r')
{
j=isstate(temp[1]);
strcpy(temp,rl[j-2].right);
dl[0]=rl[j-2].left;
dl[1]='\0';
n=strlen(temp);
for(k=0;k<2*n;k++)
pop(stack,&top);
for(m=0;dl[m]!='\0';m++)
push(stack,&top,dl[m]);
l=top;
y=stack[l-1];
isreduce(y,dl[0]);
for(m=0;temp[m]!='\0';m++)
push(stack,&top,temp[m]);
}
}
}
printt(stack,&top,inp,i);
}while(inp[i]!='\0');
if(strcmp(temp,"acc")==0)
printf(" \n accept the input ");
lse
printf(" \n do not accept the input ");
}
void push(char *s,int *sp,char item)
{
if(*sp==100)
printf(" stack is full ");
else
{
*sp=*sp+1;
s[*sp]=item;
}
}
char stacktop(char *s)
{
char i;
i=s[top];
return i;
}
oid isproduct(char x,char p)
{
int k,l;
k=ister(x);

l=isstate(p);
```

```
strcpy(temp,A[l-1].row[k-1]);
}
int ister(char x)
{
int i;
for(i=0;i<6;i++)
if(x==ter[i])
return i+1;
return 0;
}
int isnter(char x)
{
int i;
for(i=0;i<3;i++)
if(x==nter[i])
return i+1;
return 0;
}

int isstate(char p)
{
int i;
for(i=0;i<12;i++)
if(p==states[i])
return i+1;
return 0;
}
void error()
{
printf(" error in the input ");
exit(0);
}
void isreduce(char x,char p)
{
int k,l;
k=isstate(x);
l=isnter(p);

trcpy(temp,G[k-1].r[l-1]);

har pop(char *s,int *sp)
{
char item;
f(*sp==-1)
printf(" stack is empty ");
else
{
em=s[*sp];
*sp=*sp-1;


eturn item;
}
oid printt(char *t,int *p,char inp[],int i)

int r;
printf("\n");
or(r=0;r<=*p;r++)
rep(t,r);
printf("\t\t\t");
for(r=i;inp[r]!='\0';r++)
rintf("%c",inp[r]);
}
void rep(char t[],int r)
{
char c;
c=t[r];
switch(c)
{
case 'a': printf("0");
break;
case 'b': printf("1");
break;
case 'c': printf("2");
break;
case 'd': printf("3");
break;
case 'e': printf("4");
break;
case 'f': printf("5");
break;
case 'g': printf("6");
break;
case 'h': printf("7");
reak;
case 'm': printf("8");
break;
case 'j': printf("9");
break;
case 'k': printf("10");
break;
case 'l': printf("11");
break;
default :printf("%c",t[r]);
break;
}
}
```

OUTPUT:



programiz.com/c-programming/online-compiler/
Programiz C Online Compiler
Interactive C Course
main.c
Run
Output
Clear
12:20 PM

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

void push(char *,int *,char);

char stacktop(char *);

void isproduct(char,char);

int ister(char);

int isnter(char);

int isstate(char);

void error();

void isreduce(char,char);

char pop(char *,int *);

void printt(char *,int *,char [],int);
```

```
/tmp/yA3lXdk02j.o
Enter the input :i*i+i
stack            input
0           i*i+i$
0i5         *i+i$
0F3         *i+i$
0T2         *i+i$
0T2*7           i+i$
0T2*7i5         +i$
0T2*7F10            +i$
0E1         +i$
0E1+6           i$
0E1+6i5         $
0E1+6F3         $
0E1+6T9         $
0E1         $
 accept the input
```

PRACTICAL 12

AIM: Implement a C program for constructing LL (1) parsing.

CODE:

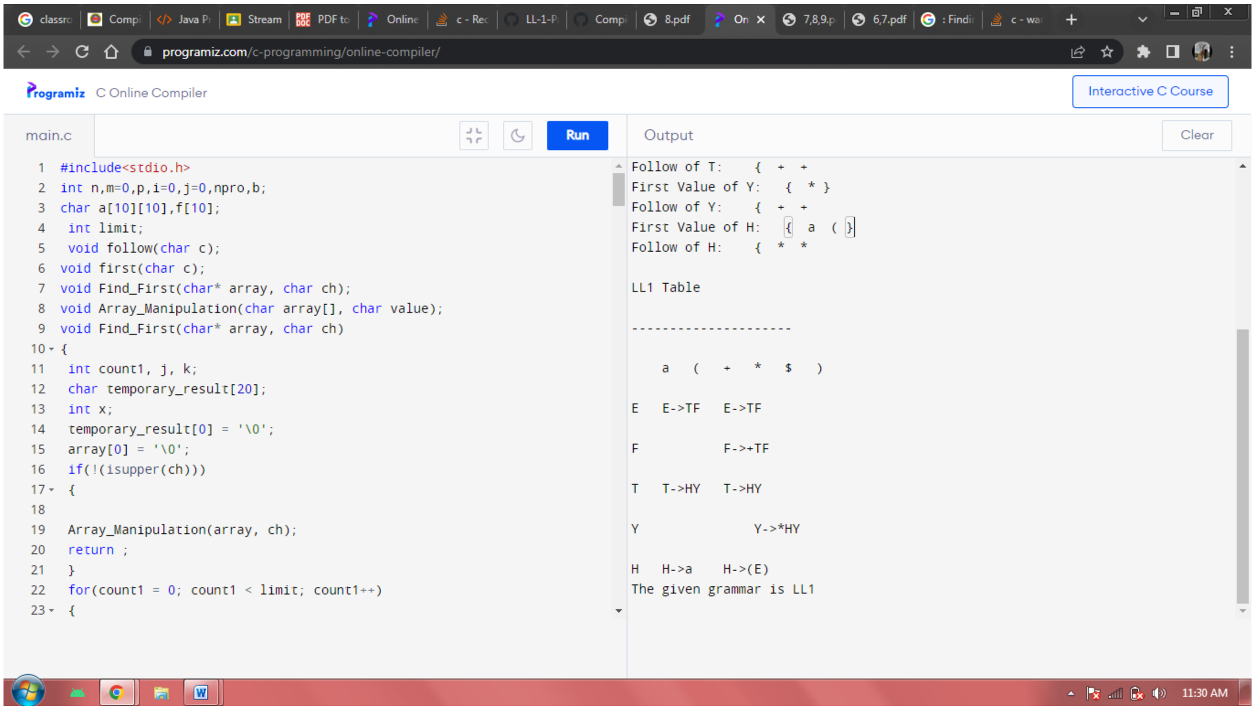
```
#include<stdio.h>
int n,m=0,p,i=0,j=0,npro,b;
char a[10][10],f[10];
 int limit;
 void follow(char c);
void first(char c);
void Find_First(char* array, char ch);
void Array_Manipulation(char array[], char value);
void Find_First(char* array, char ch)
{
 int count1, j, k;
 char temporary_result[20];
 int x;
 temporary_result[0] = '\0';
 array[0] = '\0';
 if(!(isupper(ch)))
 {

 Array_Manipulation(array, ch);
 return ;
 }
 for(count1 = 0; count1 < limit; count1++)
 {
 if(a[count1][0] == ch)
 {
 if(a[count1][3] == '^')
 {

 Array_Manipulation(array, '^');
 }
 else
 {

 j = 3;
 while(a[count1][j] != '\0')
 {
 x = 0;
 Find_First(temporary_result, a[count1][j]);
 for(k = 0; temporary_result[k] != '\0'; k++)
 {
 Array_Manipulation(array,temporary_result[k]);
 }
 for(k = 0; temporary_result[k] != '\0'; k++)
 {
 if(temporary_result[k] == '^')
 {
 x = 1;
 break;
 }
 }
 if(!x)
 {
 break;
 }
 j++;
 }
 }
 }
 }
 return;
}
void Array_Manipulation(char array[], char value)
{
 int temp;
 for(temp = 0; array[temp] != '\0'; temp++)
 {
 if(array[temp] == value)
 {
 return;
 }
 }
 array[temp] = value;
 array[temp + 1] = '\0';
}
void first(char c)
{
 int k;
 if(!isupper(c))
 f[m++]=c;
 for(k=0;k<n;k++)
 {
 if(a[k][0]==c)
{
 if(a[k][3]=='^')
 follow_fun(a[k][0]);
 else if(islower(a[k][3]))
 f[m++]=a[k][3];
 else first(a[k][3]);
}
 }
 }
 void follow_fun(char c)
 {
    if(a[0][0]==c)
 f[m++]='$';
 for(b=0;b<npro;b++)
 {
```

```
 for(j=3;j<strlen(a[b]);j++)
 {
 //printf("\nINSIDE IF for %c",c);
 if(a[b][j]==c)
{
 if(a[b][j+1]!='\0')
first(a[b][j+1]);
if(a[b][j+1]=='\0' && c!=a[b][0])
follow_fun(a[b][0]);
 }
 }
 }
 }

void main()
{
char pro[10][10],first[10][10],follow[10][10],nt[10],ter[10],res[10][10][10],temp[10];
int noter=0,nont=0,k,flag=0,count[10][10],row,col,l,index;
 char c,ch;
 char array[25];
//clrscr();
for(i=0;i<10;i++)
{
for(j=0;j<10;j++)
{
count[i][j]=NULL;
for(k=0;k<10;k++)
{
res[i][j][k]=NULL;
}
}
}
printf("Enter the no of productions:");
scanf("%d",&npro);
for(i=0;i<npro;i++)
{
//scanf("%s",pro[i]);
 scanf("%s%c",a[i],&ch);
strcpy(pro[i],a[i]);
}
limit = npro;
n = npro;
for(i=0;i<npro;i++)
{
flag=0;
for(j=0;j<nont;j++)
{
if(nt[j]==pro[i][0])
flag=1;
}
if(flag==0)
{
```

```
nt[nont]=pro[i][0];
nont++;
}
}
for(i=0;i<nont;i++)
{
m=0;
Find_First(array, nt[i] );
strcpy(first[i],array);
m=0;
 follow_fun(nt[i]);
strcpy(follow[i],f);
}
for(k=0;k<nont;k++)
{
printf("\nFirst Value of %c:\t{ ", nt[k]);
 for(i = 0; first[k][i] != '\0'; i++)
 {
 printf(" %c ", first[k][i]);
 }
 printf("}\n");

 printf("Follow of %c:\t{ ",nt[k]);
 for(i=0;i<m;i++)
printf(" %c ",follow[k][i]);
}
for(i=0;i<nont;i++)
{
flag=0;
for(j=0;j<strlen(first[i]);j++)
{
for(k=0;k<noter;k++)
{
if(ter[k]==first[i][j])
{
flag=1;
}
}
if(flag==0)
{
if(first[i][j]!='^')
{
ter[noter]=first[i][j];
noter++;
}
}
}
}
for(i=0;i<nont;i++)
{
flag=0;
for(j=0;j<strlen(follow[i]);j++)
```

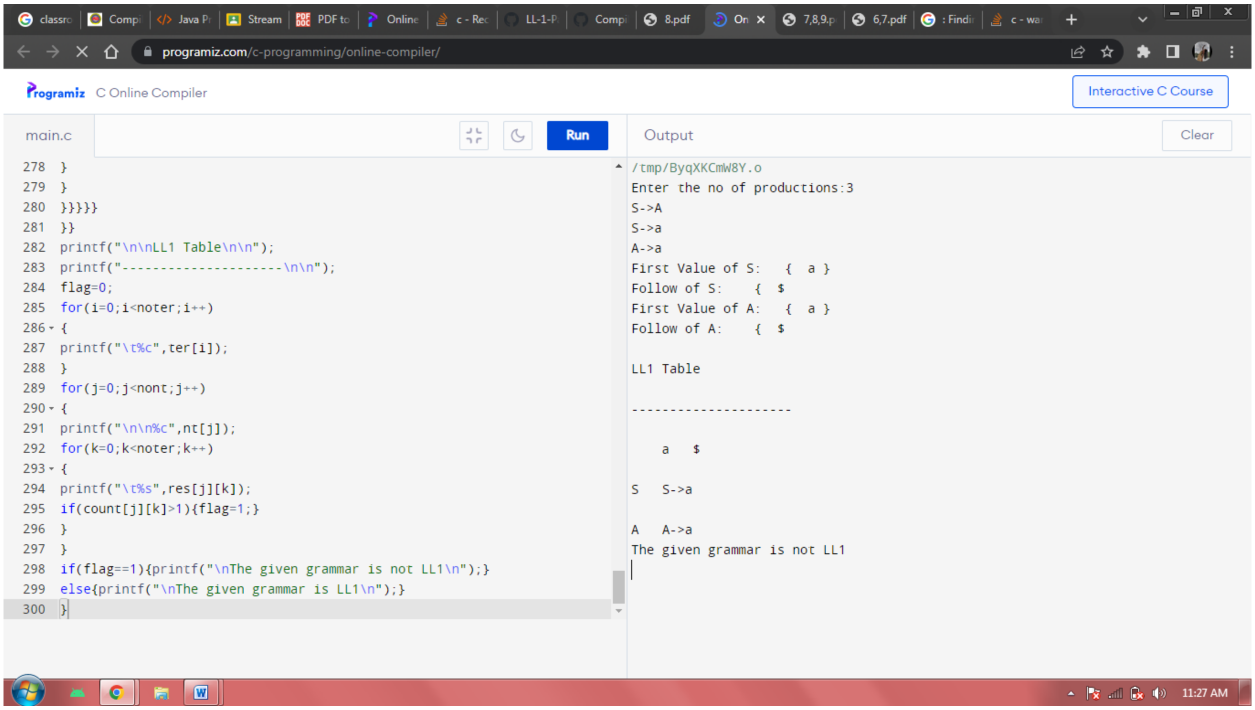
```
{
for(k=0;k<noter;k++)
{
if(ter[k]==follow[i][j])
{
flag=1;
}
}
if(flag==0)
{
ter[noter]=follow[i][j];
noter++;
}
}
}
for(i=0;i<nont;i++)
{
for(j=0;j<strlen(first[i]);j++)
{
flag=0;
if(first[i][j]=='^')
{
col=i;
for(m=0;m<strlen(follow[col]);m++)
{
for(l=0;l<noter;l++)
{
if(ter[l]==follow[col][m])
{
row=l;
}
}
temp[0]=nt[col];
temp[1]='-' ;
temp[2]='>';
temp[3]='^';
temp[4]='\0';
//printf("\ntemp %s",temp);
strcpy(res[col][row],temp);
count[col][row]+=1;
for(k=0;k<10;k++){
temp[k]=NULL; }
}
}
else{
for(l=0;l<noter;l++)
{
if(ter[l]==first[i][j])
{
row=l;
}
}
for(k=0;k<npro;k++){
if(nt[i]==pro[k][0])
{
col=i;
if((pro[k][3]==first[i][j])&&(pro[k][0]==nt[col]))
{
strcpy(res[col][row],pro[k]);
count[col][row]+=1;
}
else
{
if((isupper(pro[k][3]))&&(pro[k][0]==nt[col]))
{
flag=0;
for(m=0;m<nont;m++)
{
if(nt[m]==pro[k][3]){index=m;flag=1;}
}
if(flag==1){
for(m=0;m<strlen(first[index]);m++)
{if(first[i][j]==first[index][m])
{strcpy(res[col][row],pro[k]);
 count[col][row]+=1;}
}
}
}}}}}
}}
printf("\n\nLL1 Table\n\n");
printf("---------------------\n\n");
flag=0;
for(i=0;i<noter;i++)
{
printf("\t%c",ter[i]);
}
for(j=0;j<nont;j++)
{
printf("\n\n%c",nt[j]);
for(k=0;k<noter;k++)
{
printf("\t%s",res[j][k]);
if(count[j][k]>1){flag=1;}
}
}
if(flag==1){printf("\nThe given grammar is not LL1\n");}
else{printf("\nThe given grammar is LL1\n");}
}
```

OUTPUT:


programiz.com/c-programming/online-compiler/
Programiz C Online Compiler
Interactive C Course
main.c
Run
Output
Clear
11:30 AM

```c
#include<stdio.h>
int n,m=0,p,i=0,j=0,npro,b;
char a[10][10],f[10];
 int limit;
 void follow(char c);
void first(char c);
void Find_First(char* array, char ch);
void Array_Manipulation(char array[], char value);
void Find_First(char* array, char ch)
{
 int count1, j, k;
 char temporary_result[20];
 int x;
 temporary_result[0] = '\0';
 array[0] = '\0';
 if(!(isupper(ch)))
 {

 Array_Manipulation(array, ch);
 return ;
 }
 for(count1 = 0; count1 < limit; count1++)
 {
```

```
Follow of T:    {  +  +
First Value of Y:   {  * }
Follow of Y:    {  +  +
First Value of H:   {  a  ( }
Follow of H:    {  *  *

LL1 Table

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


    a   (   +   *   $   )

E   E->TF   E->TF

F           F->+TF

T   T->HY   T->HY

Y               Y->*HY

H   H->a    H->(E)
The given grammar is LL1
```


programiz.com/c-programming/online-compiler/
Programiz C Online Compiler
Interactive C Course
main.c
Run
Output
Clear
11:27 AM

```c
}
}
}}}}}
}}
printf("\n\nLL1 Table\n\n");
printf("--------------------\n\n");
flag=0;
for(i=0;i<noter;i++)
{
printf("\t%c",ter[i]);
}
for(j=0;j<nont;j++)
{
printf("\n\n%c",nt[j]);
for(k=0;k<noter;k++)
{
printf("\t%s",res[j][k]);
if(count[j][k]>1){flag=1;}
}
}
if(flag==1){printf("\nThe given grammar is not LL1\n");}
else{printf("\nThe given grammar is LL1\n");}
}
```

```
/tmp/ByqXKCmW8Y.o
Enter the no of productions:3
S->A
S->a
A->a
First Value of S:   {  a }
Follow of S:    {  $
First Value of A:   {  a }
Follow of A:    {  $

LL1 Table

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


    a   $

S   S->a

A   A->a
The given grammar is not LL1
```