# PRACTICAL: 1

# Practical: 1

**Aim: Implementation of finite automata and string validation.**

```
#include<stdio.h>
#define max 100
int main()
{
	char str[max],f='a';
	int i;
	printf("Enter string: ");
	scanf("%s",str);
	for(i=0;str[i]!='\0';i++)
	{
		switch(f)
		{
			case 'a': if(str[i]=='0')
			{
				f='b';
			}
			else if(str[i]=='1')
			{
				f='a';
			}
			break;

			case 'b': if(str[i]=='0')
			{
				f='b';
```

```
                }
                else if(str[i]=='1')
                {
                        f='c';
                }
                break;

                case 'c': if(str[i]=='0')
                {
                        f='b';
                }
                else if(str[i]=='1')
                {
                        f='a';
                }
                break;
            }
        }
        if(f=='c')
        {
            printf("String Accepted..!");
        }
        else
        {
            printf("String Not Accepted..!");
        }
    }
```

**Output:**

```
Enter string: 10011001
String Accepted..!

...Program finished with exit code 0
Press ENTER to exit console.
```

```
Enter string: 0110
String Not Accepted..!

...Program finished with exit code 0
Press ENTER to exit console.
```

# PRACTICAL: 2

# Practical: 2

**Aim: Introduction to Lex Tool.**

## Lex

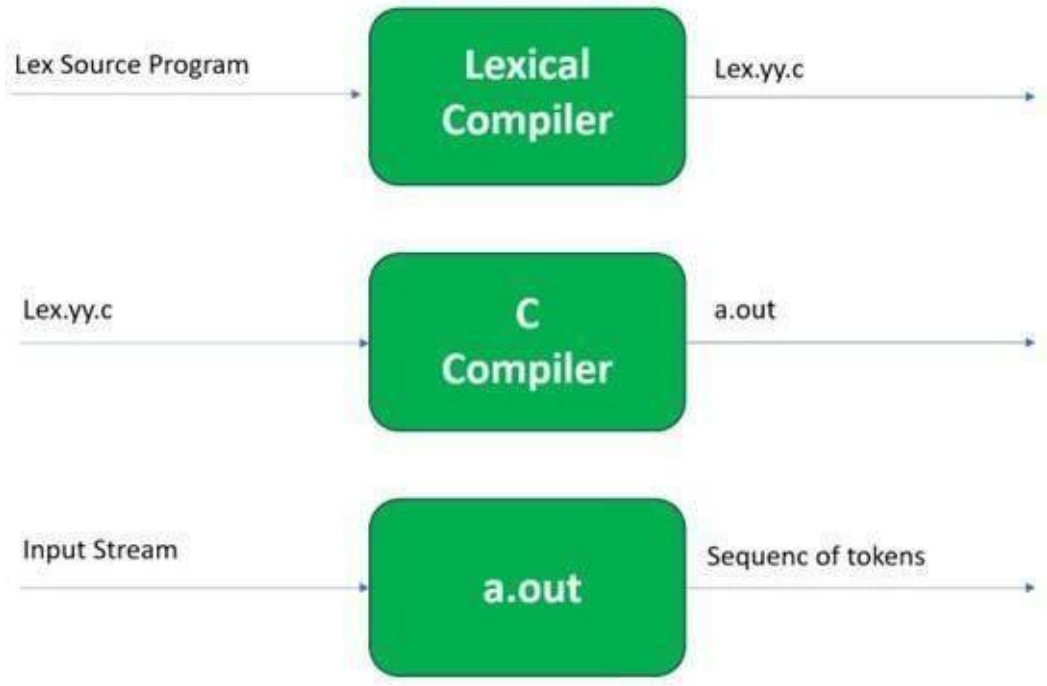
Lex is a tool or a computer program that generates Lexical Analyzers (converts the stream of characters into tokens). The Lex tool itself is a compiler. The Lex compiler takes the input and transforms that input into input patterns. It is commonly used with YACC(Yet Another Compiler Compiler). It was written by Mike Lesk and Eric Schmidt.

## Function of Lex

1. In the first step the source code which is in the Lex language having the file name 'File.l' gives as input to the Lex Compiler commonly known as Lex to get the output as lex.yy.c.

2. After that, the output lex.yy.c will be used as input to the C compiler which gives the output in the form of an 'a.out' file, and finally, the output file a.out will take the stream of character and generates tokens as output.

lex.yy.c: It is a C program.
File.l: It is a Lex source program
a.out: It is a Lexical analyzer



## Lex File Format

A Lex program consists of three parts and is separated by %% delimiters:-

Declarations
%%
Translation rules
%%
Auxiliary procedures

**Declarations:** The declarations include declarations of variables.

**Transition rules:** These rules consist of Pattern and Action.

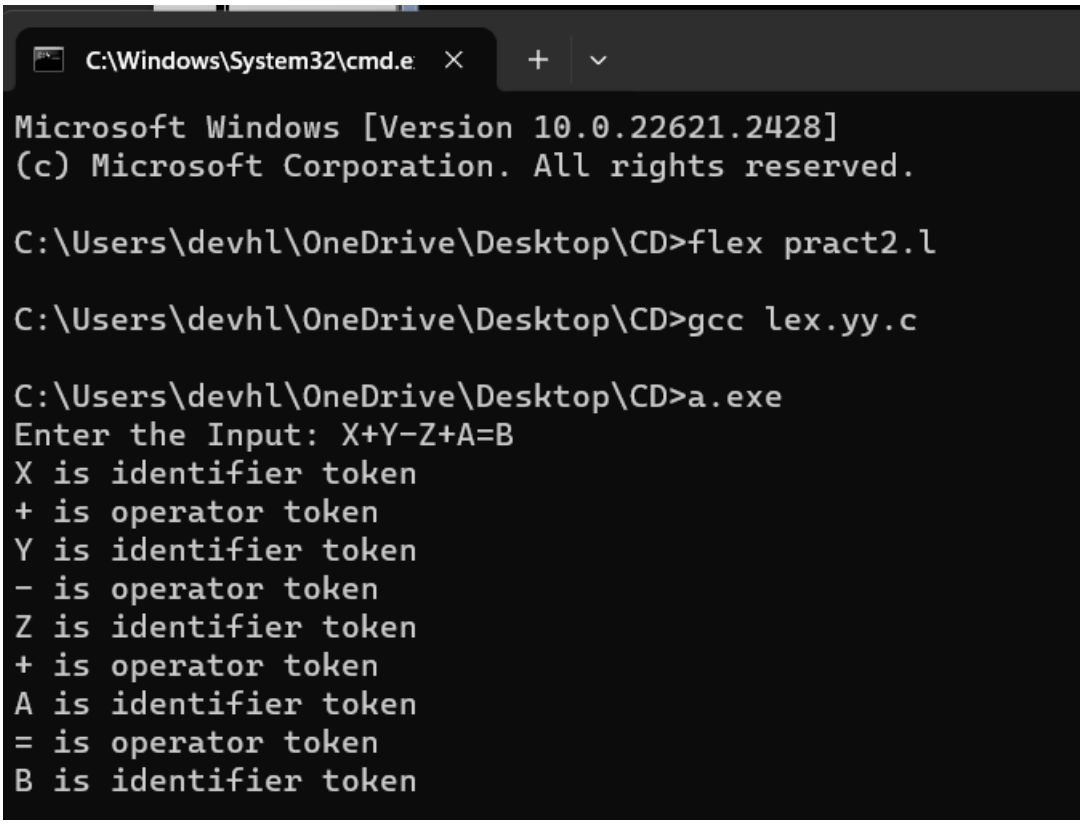
**Auxiliary procedures:** The Auxilary section holds auxiliary functions used in the actions.

## Code :

```
%{

        #include<stdio.h>

%}

Letter [A-Za-z]

Digit [0-9]

Id      {Letter}({Letter}|{Digit})*

Op "+"|"-"|"<"|"<="|">="|"*"|"="

%%

{Id} {printf("%s is identifier token \n",yytext);}

{Digit}+ {printf("%s is number token \n",yytext);}

{Op} {printf("%s is operator token \n",yytext);}

%%

int main()

{

        printf("Enter the Input: ");

        yylex();
```

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

**Output:**



# PRACTICAL: 3

# Practical: 3

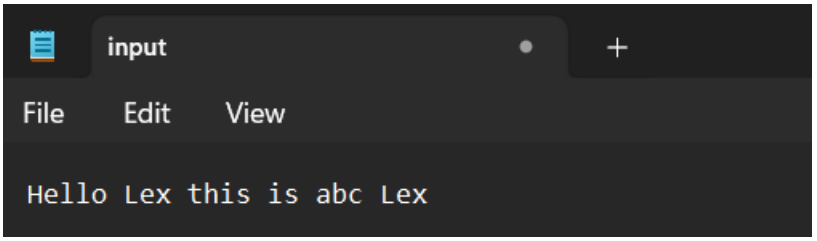
**Aim: Implement following Programs Using Lex.**

# Practical: 3(a)

**Aim: Generate Histogram of words.**

```
%{
        #include<stdio.h>
        #include<string.h>
        char word[]="Lex";
        int count=0;
%}
%%
[a-zA-Z]+ {if(strcmp(yytext,word)==0)
count++;}
. ;
%%
int yywrap()
{
        return 1;
}
int main()
{
        extern FILE*yyin,*yyout;
        yyin=fopen("input.txt","r");
        yylex();
        printf("Total Number of Lex = %d",count);
}
```

**Output:**




```
C:\Users\devhl\OneDrive\Desktop\CD>flex pr3a.l

C:\Users\devhl\OneDrive\Desktop\CD>gcc lex.yy.c

C:\Users\devhl\OneDrive\Desktop\CD>a.exe
Total Number of Lex = 2
C:\Users\devhl\OneDrive\Desktop\CD>
```

# Practical: 3(b)

**Aim: Ceasor Cypher.**

```
%%
[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()
{
        printf("Plain Text Is:");
        yylex();
}
int yywrap(void)
{
        return 1;
}
```

**Output:**

```
C:\Users\devhl\OneDrive\Desktop\CD>flex pr3b.l

C:\Users\devhl\OneDrive\Desktop\CD>gcc lex.yy.c

C:\Users\devhl\OneDrive\Desktop\CD>a.exe
Plain Text Is:Hello Lex
Khoor Oha
```

# Practical: 3(c)

**Aim: Extract single and multiline comments from C Program.**

## (.l File)

```
%{
#include<stdio.h>
%}
%%
"/*"[a-zA-z0-9' '\t\n]*"*/" {};
"//"[a-zA-Z0-9' '\t]* {};
%%
int yywrap()
{
	return 1;
}
int main()
{
	yyin=fopen("file.c","r");
	yyout=fopen("out.c","w");
	yylex();
	return 0;
}
```

## (.C File)

```
//Addition of two numbers..
#include <stdio.h>
int main() {
	int num1, num2, sum;
```

```
        printf("Enter first number:");
        scanf("%d" , &num1);
        printf("Enter second num:");
        scanf("%d" , &num2);
        /*Add num1
                &
            num2*/
        sum = num1 + num2;
        printf("The sum of %d and % is %d.", num1,num2,sum);
        return 0;
}
```

**Output:**

```
C:\Users\devhl\OneDrive\Desktop\CD>flex pr3c.l

C:\Users\devhl\OneDrive\Desktop\CD>gcc lex.yy.c

C:\Users\devhl\OneDrive\Desktop\CD>a.exe

C:\Users\devhl\OneDrive\Desktop\CD>
```

out.c

```c
#include <stdio.h>
int main() {
int num1, num2, sum;
210490131501 Compiler Design
SNPITRC/CSE/2023-24/SEM-7/3170701 P a g e | 14
printf("Enter first number:");
scanf("%d" , &num1);
printf("Enter second num:");
scanf("%d" , &num2);
sum = num1 + num2;
printf("The sum of %d and % is %d.", num1,num2,sum);
return 0;
}
```

Remove Comments: (out.c)

# PRACTICAL: 4

# Practical: 4

**Aim: Implement following Programs Using Lex.**

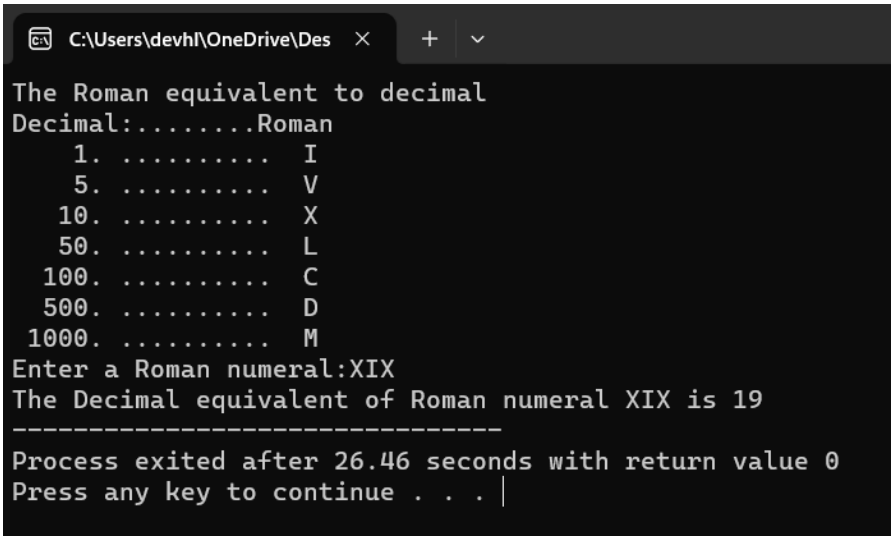
# Practical: 4(a)

**Aim: Convert Roman to Decimal.**

```
#include  <stdio.h>
#include <conio.h>
main(){
   char roman[30];
   int deci=0;
   int length,i,d[30];
   printf("The Roman equivalent to decimal\n");
   printf("Decimal: ..... Roman\n");
   printf("%5d. .....%3c\n",1,'I');
   printf("%5d. .....%3c\n",5,'V');
   printf("%5d. .....%3c\n",10,'X');
   printf("%5d. .....%3c\n",50,'L');
   printf("%5d. .....%3c\n",100,'C');
   printf("%5d. .....%3c\n",500,'D');
   printf("%5d. .....%3c\n",1000,'M');
   printf("Enter a Roman numeral:");
   scanf("%s",roman);
   length=strlen(roman);
   for(i=0;i<length;i++){
            switch(roman[i]){
                     case 'm':
                     case 'M': d[i]=1000; break;
                     case 'd':
                     case 'D': d[i]= 500; break;
                     case 'c':
                     case 'C': d[i]= 100; break;
                     case 'l':
                     case 'L': d[i]= 50; break;
                     case 'x':
                     case 'X': d[i]= 10; break;;
                     case 'v':
                     case 'V': d[i]= 5; break;
                     case 'i':
                     case 'I': d[i]= 1;
                  }
}
for(i=0;i<length;i++){
   if(i==length-1 || d[i]>=d[i+1])
```

```
            deci += d[i];
            else
                    deci -= d[i];
            }
    printf("The Decimal equivalent of Roman numeral %s is %d", roman, deci);
}
```

**Output:**



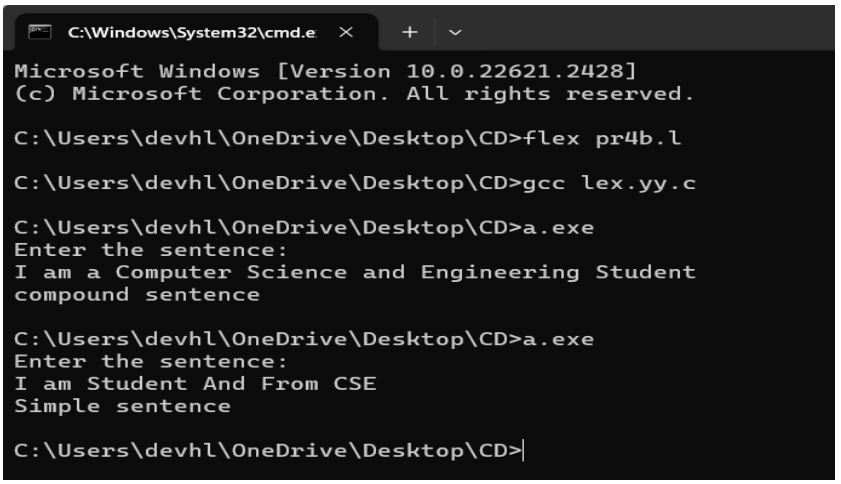

# Practical: 4(b)

**Aim: Check weather given statement is compound or simple.**

```
%{

#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:**

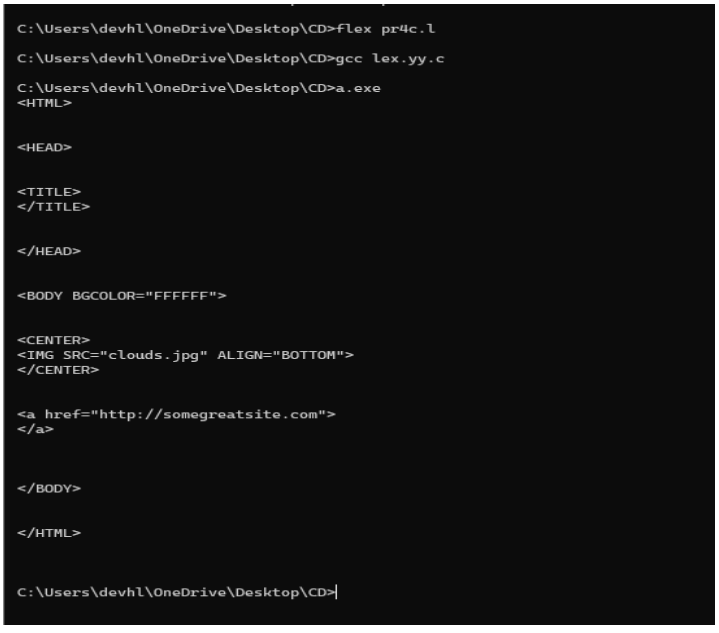


# Practical: 4(c)

**Aim: Extract html tags from .html file.**

```
%{
#include<stdio.h>
%}
%%
\<[^>]*\> fprintf(yyout,"%s\n",yytext);
.|\n;
%%
int yywrap()
{
	return 1;
}
int main()
{
	yyin=fopen("tags.html","r"); yylex();
	return 0;
}
```

**Output:**



# PRACTICAL: 5

# Practical: 5

**Aim: Implementation of Recursive Descent Parser without backtracking**

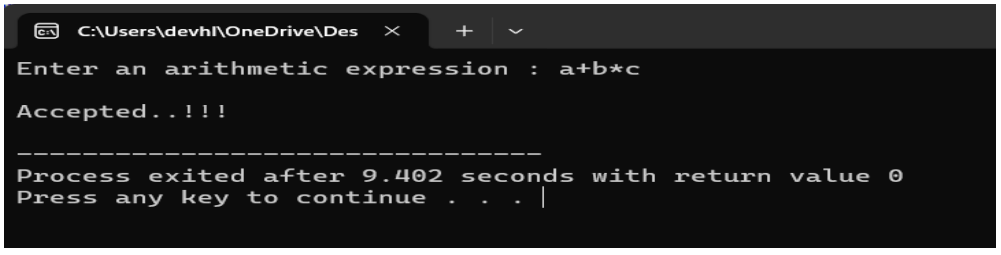
**Program :**

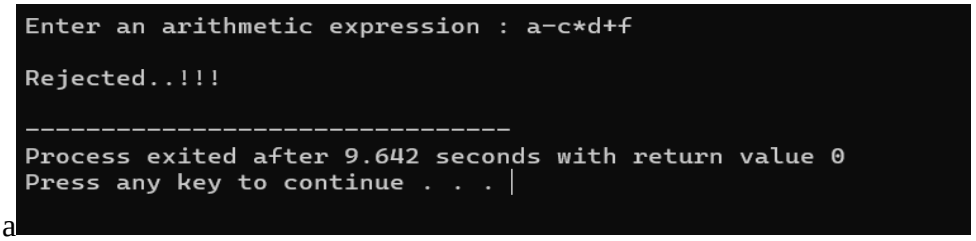
Program :

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>
char input[10];
int i,error;
void E();
void T();
void Eprime();
void Tprime();
void F();
 void main()
 {
        i=0;
        error=0;
        printf("Enter an arithmetic expression : "); // Eg: a+a*a
        gets(input);
        E();
        if(strlen(input)==i&&error==0)
        printf("\nAccepted..!!!\n");
        else printf("\nRejected..!!!\n");
 }
void E()
{
        T();
        Eprime();
}
void Eprime()
{
        if(input[i]=='+')
        {
                i++;
```

```
                T();
                Eprime();
        }
 }
void T()
{
        F();
        Tprime();
}
void Tprime()
{
         if(input[i]=='*')
        {
                i++;
                F();
                Tprime();
        }
 }
 void F()
{
        if(isalnum(input[i]))i++;
        else if(input[i]=='(')
        {
                 i++;
                E();
                if(input[i]==')')
                 i++;
                else error=1;
        }
        else error=1;
 }
```

**Output:**




a

# PRACTICAL: 6

# Practical: 6

**Aim: Finding "First" set**

**Input: The string consists of grammar symbols.**

**Output: The First set for a given string.**

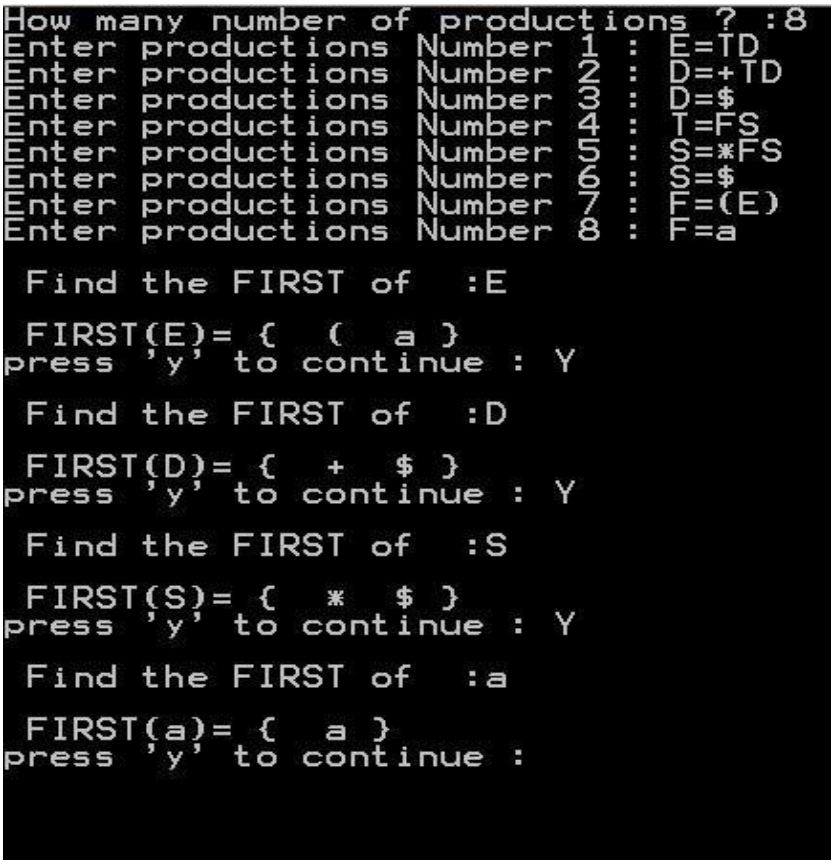
```
#include<stdio.h>
#include<ctype.h>
void FIRST(char[],char );
void
addToResultSet(char[],char);int
numOfProductions;
char productionSet[10][10];
main()
{
   int i;
   char choice;
   char c;
   char result[20];
   printf("How many number of productions :");
   scanf(" %d",&numOfProductions);
   for(i=0;i<numOfProductions;i++)//
   {
      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
        resultprintf("}\n");
         printf("press 'y' to continue :
        ");scanf(" %c",&choice);
    }
    while(choice=='y'||choice =='Y');
}
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(Yi1).
        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;
               }
             //No ε found, no need to check next element
             if(!foundEpsilon)
                break;
                j++;
           }
        }
    }
}
   return ;
}

/* addToResultSet adds the computed
 *element to result set.
 *This code avoids multiple inclusion of elements*/
```

```
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';
}
```

**Output:**



# PRACTICAL: 7

# Practical: 7

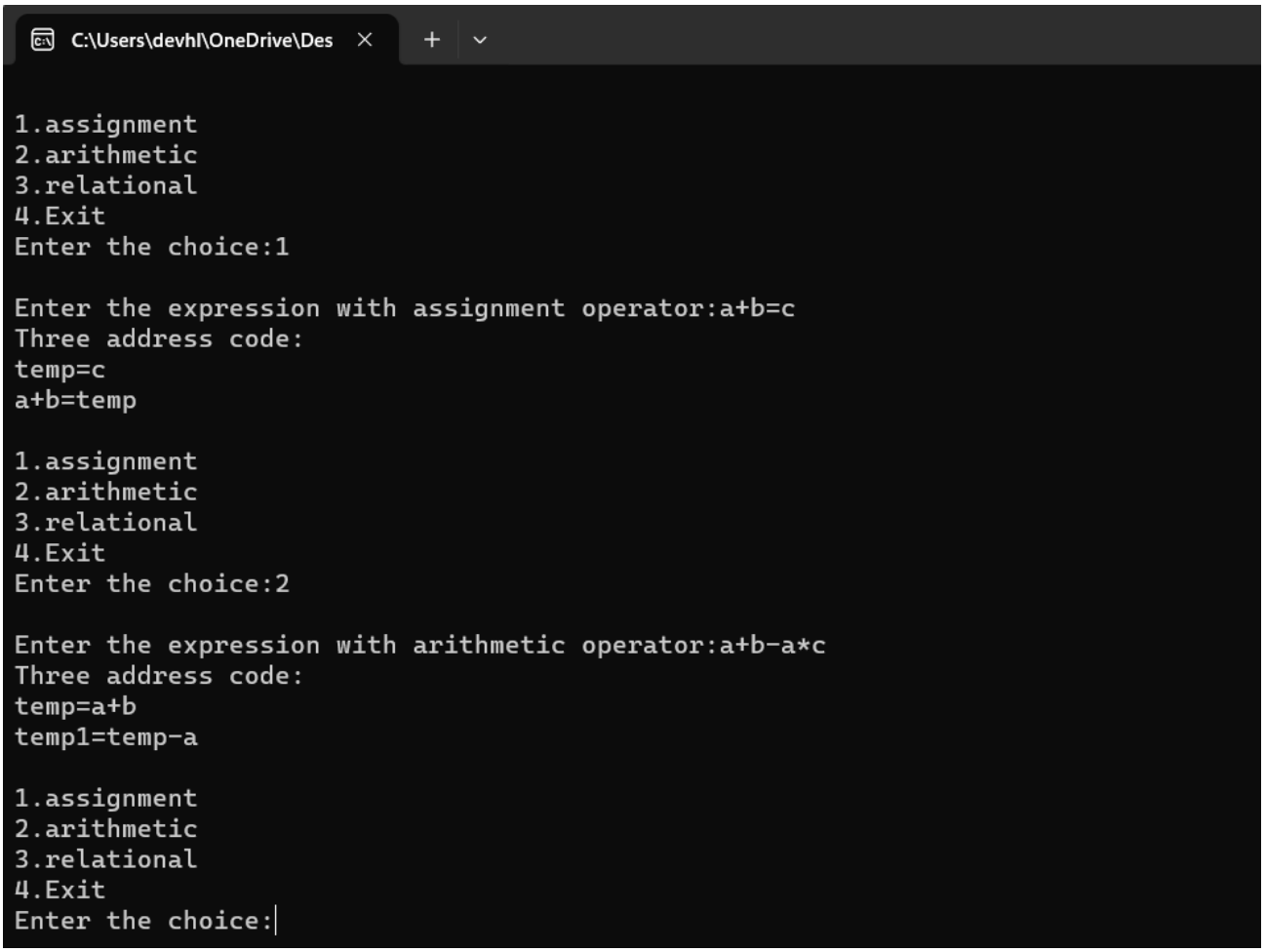
**Aim: Generate 3-tuple intermediate code for given infix expression**

```
#include<stdio.h>
#include<string.h>
void pm();
void plus();
void div();
int i,ch,j,l,addr=100;
char ex[10], exp[10] ,exp1[10],exp2[10],id1[5],op[5],id2[5];
void main()
{
        while(1)
        {
                printf("\n1.assignment\n2.arithmetic\n3.relational\n4.Exit\nEnter the choice:");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1:
                                printf("\nEnter the expression with assignment operator:");
                                scanf("%s",exp);
                                l=strlen(exp);
                                exp2[0]='\0';
                                i=0;
                                while(exp[i]!='=')
                                {
                                        i++;
                                }
                                strncat(exp2,exp,i);
                                strrev(exp);
                                exp1[0]='\0';
                                strncat(exp1,exp,l-(i+1));
                                strrev(exp1);
                                printf("Three address code:\ntemp=%s\n%s=temp\n",exp1,exp2);
                                break;
                        case 2:
                                printf("\nEnter the expression with arithmetic operator:");
                                scanf("%s",ex);
                                strcpy(exp,ex);
                                l=strlen(exp);
                                exp1[0]='\0';
                                for(i=0;i<l;i++)
                                {
                                        if(exp[i]=='+'||exp[i]=='-')
                                        {
                                                if(exp[i+2]=='/'||exp[i+2]=='*')
                                                {
                                                        pm();
                                                        break;
                                                }
```

```
                        else
                        {
                                plus();
                                break;
                        }
                }
                else if(exp[i]=='/'||exp[i]=='*')
                {
                        div();
                        break;
                }
        }
        break;
case 3:
        printf("Enter the expression with relational operator");
        scanf("%s%s%s",&id1,&op,&id2);
        if(((strcmp(op,"<")==0)||(strcmp(op,">")==0)||(strcmp(op,"<=")==0)
        ||(strcmp(op,">=")==0)||(strcmp(op,"==")==0)||(strcmp(op,"!=")==0)
        )==0)
                printf("Expression is error");
        else
        {
                printf("\n%d\tif %s%s%s goto %d",addr,id1,op,id2,addr+3);
                addr++;
                printf("\n%d\t T:=0",addr);
                addr++;
                printf("\n%d\t goto %d",addr,addr+2);
                addr++;
                printf("\n%d\t T:=1",addr);
        }
        break;
case 4:
        exit(0);
        }
        }
        }
        void pm()
        {
                strrev(exp);
                j=l-i-1;
                strncat(exp1,exp,j);
                strrev(exp1);
                printf("Three address
        code:\ntemp=%s\ntemp1=%c%ctemp\n",exp1,exp[j+1],exp[j]);
        }
        void div()
        {
                strncat(exp1,exp,i+2);
                printf("Three address
        code:\ntemp=%s\ntemp1=temp%c%c\n",exp1,exp[i+2],exp[i+3]);
        }
        void plus()
```

```
{
        strncat(exp1,exp,i+2);
         printf("Three address
code:\ntemp=%s\ntemp1=temp%c%c\n",exp1,exp[i+2],exp[i+3]);
}
```

**Output:**



# PRACTICAL: 8

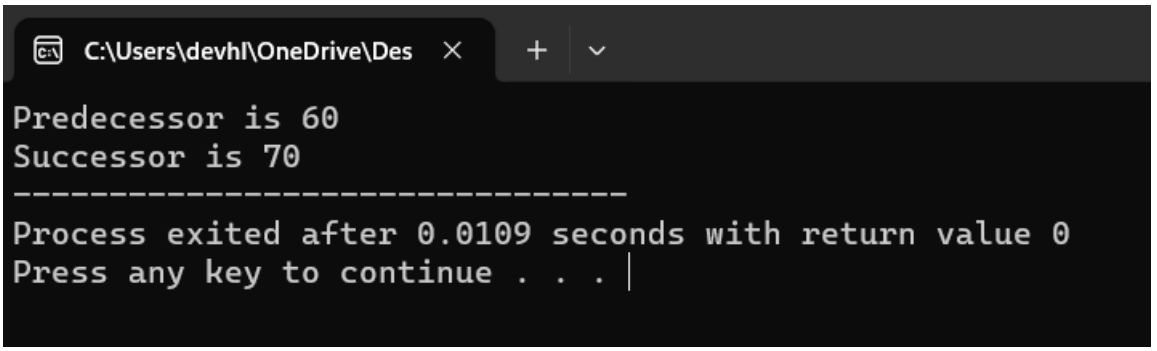
# Practical: 8

**Aim: Extract Predecessor and Successor from given Control Flow Graph**
**// 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)
		{
			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) ;
	}
```

```
}
// A utility function to create a new BST node
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
        /* Let us create following BST

          50
          /  \
         30    70
        /   \   / \
       20  40 60 80 */

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:\Users\devhl\OneDrive\Des
Predecessor is 60
Successor is 70
--------------------------------
Process exited after 0.0109 seconds with return value 0
Press any key to continue . . .

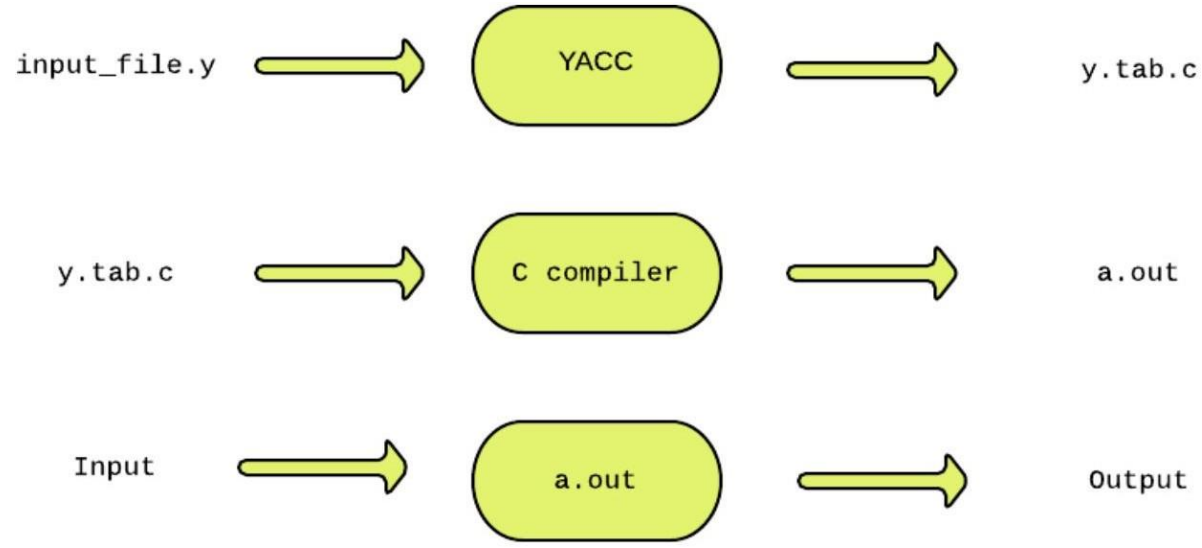
# PRACTICAL: 9

# Practical: 9

**Aim: Introduction to YACC and generate Calculator Program**

**What is YACC?**

YACC stands for yet another 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. It takes input as a CFG file (file.y) and outputs a parser file (y.tab.c).




**Program:**

**➢ LEX Program :**

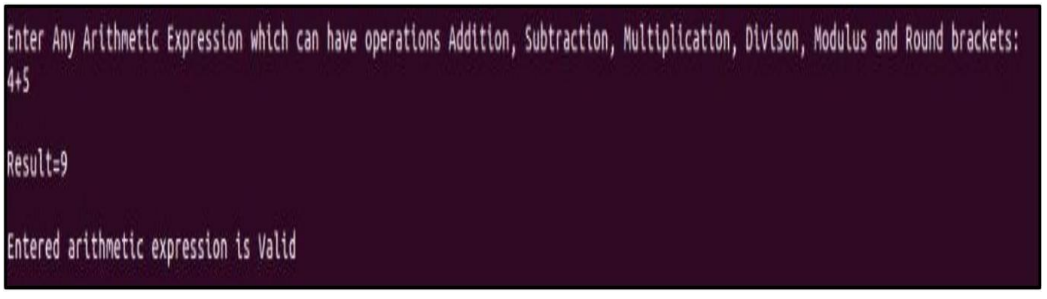
```
%{
/* Definition section */ #include<stdio.h>
 #include "y.tab.h"extern int yylval;
%}
/* Rule Section */
%%
[0-9]+ { yylval=atoi(yytext); return NUMBER; }
[\t] ;
[\n] return 0;
. return yytext[0];
%%
int yywrap()
{
        return 1;
}
```

**YACC Program :**

```
%{
```

```
/* Definition section */
#include<stdio.h> int flag=0;
%}
%token NUMBER
%left '+' '-'
%left '*' '/' '%'
%left '(' ')'
/* Rule Section */
%%
ArithmeticExpression: E{
printf("Result=%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()
{
yyparse();
}
void yyerror()
{
printf("\nEntered arithmetic expression is
Invalid\n\n"); flag=1;
}
```

**Output:**



# PRACTICAL: 10

# Practical: 10

**Aim: Finding "Follow" set**

**Input: The string consists of grammar symbols.**

**Output: The Follow set for a given string.**

Program:

```
// C program to calculate the First and
// Follow sets of a given grammar
#include<stdio.h>
#include<ctype.h>
#include<string.h>
// Functions to calculate Follow
void followfirst(char, int, int);
void follow(char c);
// Function to calculate First
void findfirst(char, int, int);
int count, n = 0;
// Stores the final result
// of the First Sets
char calc_first[10][100];
// Stores the final result
// of the Follow Sets
char calc_follow[10][100];
int m = 0;
// Stores the production rules
char production[10][10];
char f[10], first[10];
int k;
char ck;
int e;
int main(int argc, char **argv)
{
int jm = 0;
int km = 0;
int i, choice;
char c, ch;
```

```
count = 8;
// The Input grammar
strcpy(production[0], "E=TR");
strcpy(production[1], "R=+TR");
strcpy(production[2], "R=#");
strcpy(production[3], "T=FY");
strcpy(production[4], "Y=*FY");
strcpy(production[5], "Y=#");
strcpy(production[6], "F=(E)");
strcpy(production[7], "F=i");
int kay;
char done[count];
int ptr = -1;
// Initializing the calc_first array
for(k = 0; k < count; k++) {
for(kay = 0; kay < 100; kay++) {
calc_first[k][kay] = '!';
}
}
int point1 = 0, point2, xxx;
for(k = 0; k < count; k++)
{
c = production[k][0];
point2 = 0;
xxx = 0;
// Checking if First of c has
// already been calculated
for(kay = 0; kay <= ptr; kay++)
if(c == done[kay])
xxx = 1;
if (xxx == 1)
continue;
// Function call
findfirst(c, 0, 0);
ptr += 1;
// Adding c to the calculated list
done[ptr] = c;
printf("\n First(%c) = { ", c);
```

```
calc_first[point1][point2++] = c;
// Printing the First Sets of the grammar
for(i = 0 + jm; i < n; i++) {
int lark = 0, chk = 0;
for(lark = 0; lark < point2; lark++) {
if (first[i] == calc_first[point1][lark])
{
chk = 1;
break;
}
}
if(chk == 0)
{
printf("%c, ", first[i]);
calc_first[point1][point2++] = first[i];
}
}
printf("}\n");
jm = n;
point1++;
}
printf("\n");
printf("----------\n\n");
char donee[count];
ptr = -1;
// Initializing the calc_follow array
for(k = 0; k < count; k++) {
for(kay = 0; kay < 100; kay++) {
calc_follow[k][kay] = '!';
}
}
point1 = 0;
int land = 0;
for(e = 0; e < count; e++)
{
ck = production[e][0];
point2 = 0;
xxx = 0;
```

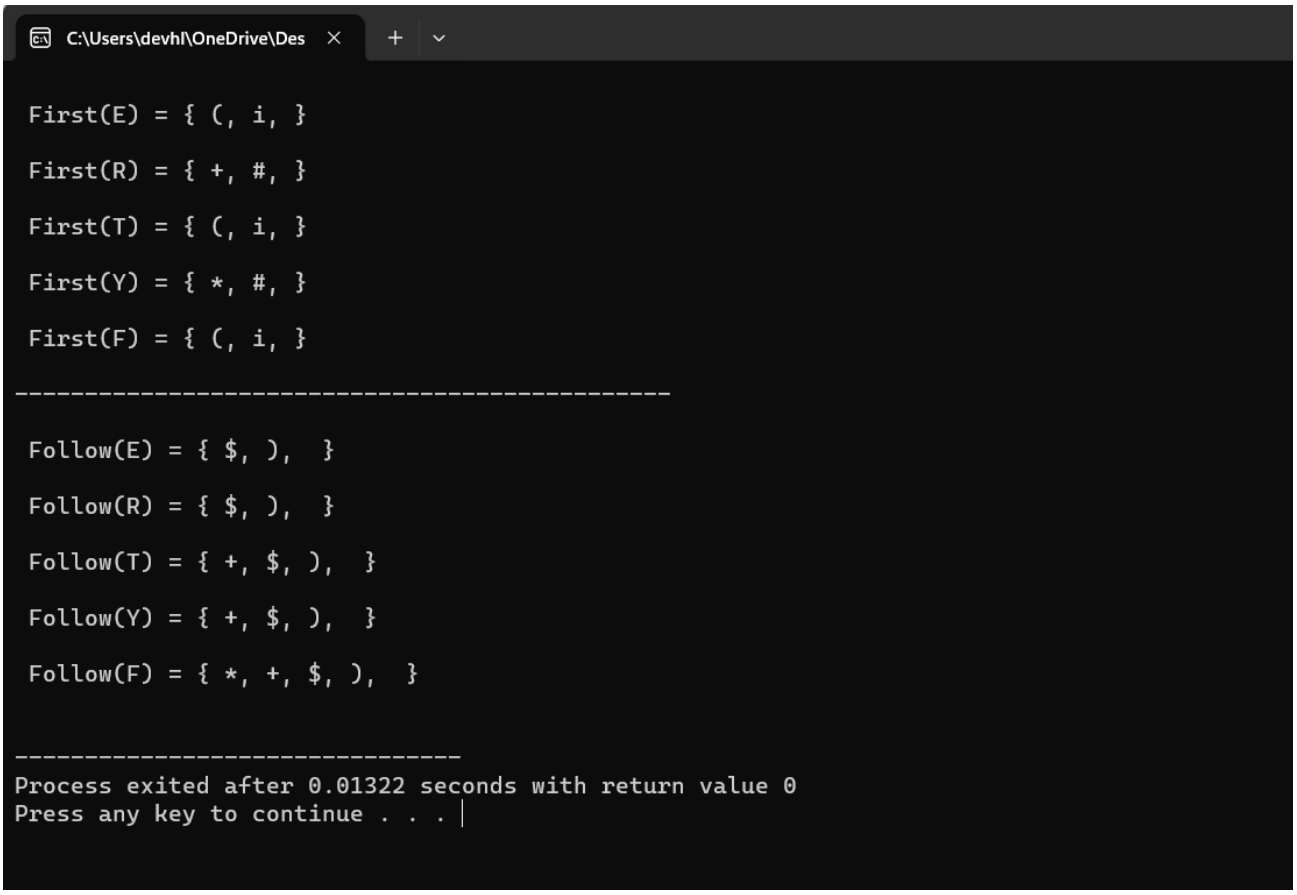
```
// Checking if Follow of ck
// has already been calculated
for(kay = 0; kay <= ptr; kay++)
if(ck == donee[kay])
xxx = 1;
if (xxx == 1)
continue;
land += 1;
// Function call
follow(ck);
ptr += 1;
// Adding ck to the calculated list
donee[ptr] = ck;
printf(" Follow(%c) = { ", ck);
calc_follow[point1][point2++] = ck;
// Printing the Follow Sets of the grammar
for(i = 0 + km; i < m; i++) {
int lark = 0, chk = 0;
for(lark = 0; lark < point2; lark++)
{
if (f[i] == calc_follow[point1][lark])
{
chk = 1;
break;
}
}
if(chk == 0)
{
printf("%c, ", f[i]);
calc_follow[point1][point2++] = f[i];
}
}
printf(" }\n\n");
km = m;
point1++;
}
}
void follow(char c)
```

```
{
int i, j;
// Adding "$" to the follow
// set of the start symbol
if(production[0][0] == c) {
f[m++] = '$';
}
for(i = 0; i < 10; i++)
{
for(j = 2;j < 10; j++)
{
if(production[i][j] == c)
{
if(production[i][j+1] != '\0')
{
// Calculate the first of the next
// Non-Terminal in the production
followfirst(production[i][j+1], i, (j+2));
}
if(production[i][j+1]=='\0' && c!=production[i][0])
{
// Calculate the follow of the Non-Terminal
// in the L.H.S. of the production
follow(production[i][0]);
}
}
}
}
}
void findfirst(char c, int q1, int q2)
{
int j;
// The case where we
// encounter a Terminal
if(!(isupper(c))) {
first[n++] = c;
}
for(j = 0; j < count; j++)
```

```
{
if(production[j][0] == c)
{
if(production[j][2] == '#')
{
if(production[q1][q2] == '\0')
first[n++] = '#';
else if(production[q1][q2] != '\0'
&& (q1 != 0 || q2 != 0))
{
// Recursion to calculate First of New
// Non-Terminal we encounter after epsilon
findfirst(production[q1][q2], q1, (q2+1));
}
else
first[n++] = '#';
}
else if(!isupper(production[j][2]))
{
first[n++] = production[j][2];
}
else
{
// Recursion to calculate First of
// New Non-Terminal we encounter
// at the beginning
findfirst(production[j][2], j, 3);
}
}
}
}
void followfirst(char c, int c1, int c2)
{
int k;
// The case where we encounter
// a Terminal
if(!(isupper(c)))
f[m++] = c;
```

```
else
{
int i = 0, j = 1;
for(i = 0; i < count; i++)
{
if(calc_first[i][0] == c)
break;
}
//Including the First set of the
// Non-Terminal in the Follow of
// the original query
while(calc_first[i][j] != '!')
{
if(calc_first[i][j] != '#')
{
f[m++] = calc_first[i][j];
}
else
{
if(production[c1][c2] == '\0')
{
// Case where we reach the
// end of a production
follow(production[c1][0]);
}
else
{
// Recursion to the next symbol
// in case we encounter a "#"
followfirst(production[c1][c2], c1, c2+1);
}
}
j++;
}
}
}
```

**Output:**


C:\Users\devhl\OneDrive\Des
First(E) = { (, i, }
First(R) = { +, #, }
First(T) = { (, i, }
First(Y) = { *, #, }
First(F) = { (, i, }
Follow(E) = { $, ), }
Follow(R) = { $, ), }
Follow(T) = { +, $, ), }
Follow(Y) = { +, $, ), }
Follow(F) = { *, +, $, ), }
Process exited after 0.01322 seconds with return value 0
Press any key to continue . . .

# PRACTICAL: 11

# Practical: 11

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

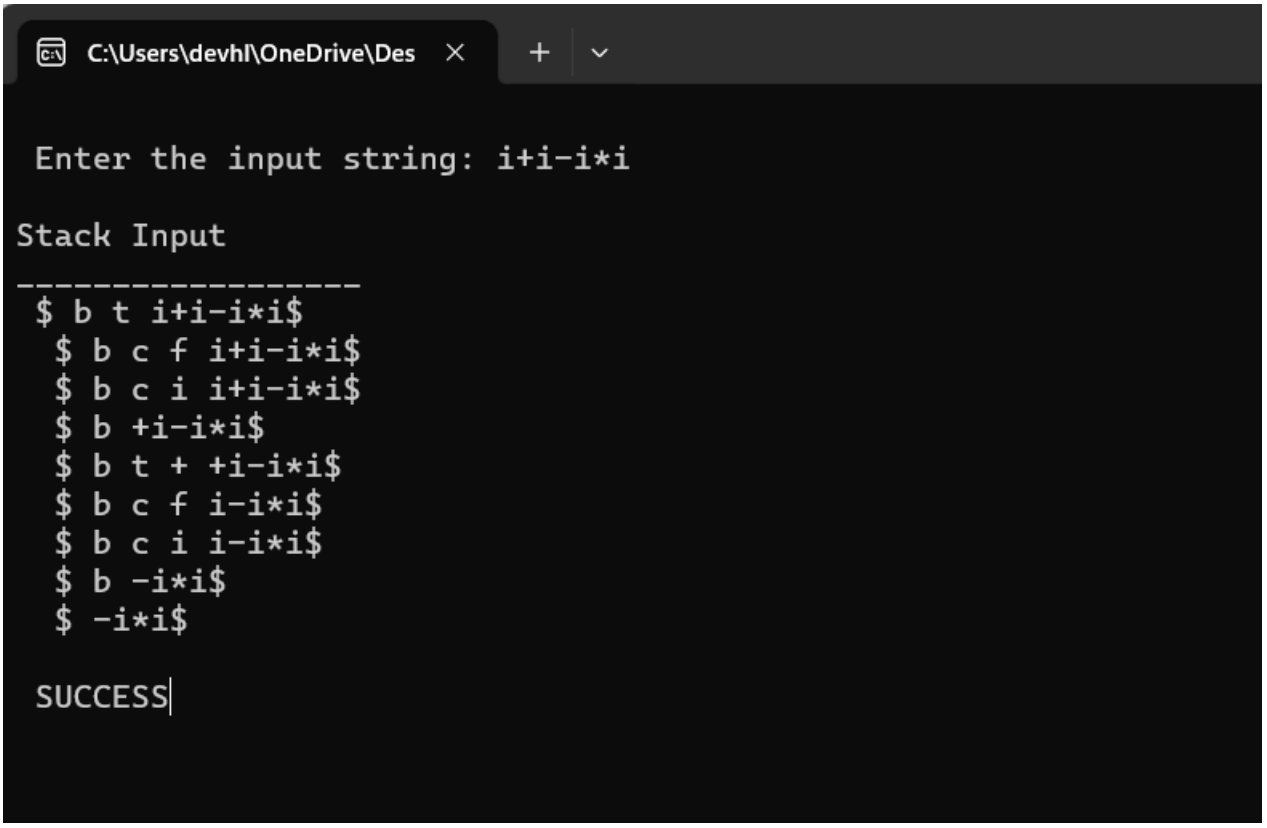
```
#include<stdio.h>
#include<conio.h>
#include<string.h>
char s[20],stack[20];
void main()
{
char m[5][6][3]={"tb"," "," ","tb"," "," "," ","+tb"," "," ","n","n","fc"," "," ","fc"," "," "," "
","n","*fc","a","n","n","i"," "," ","(e)"," "," "};
int size[5][6]={2,0,0,2,0,0,0,3,0,0,1,1,2,0,0,2,0,0,0,1,3,0,1,1,1,0,0,3,0,0};
int i,j,k,n,str1,str2;
printf("\n Enter the input string: ");
scanf("%s",s);
strcat(s,"$");
n=strlen(s);
stack[0]='$';
stack[1]='e';
i=1;
j=0;
printf("\nStack Input\n");
printf("_____\n");
while((stack[i]!='$')&&(s[j]!='$'))
{
if(stack[i]==s[j])
{
i--;
```

```
j++;
}
switch(stack[i])
{
case 'e': str1=0;
break;
case 'b': str1=1;
break;
case 't': str1=2;
break;
case 'c': str1=3;
break;
case 'f': str1=4;
break;
}
switch(s[j])
{
case 'i': str2=0;
break;
case '+': str2=1;
break;
case '*': str2=2;
break;
case '(': str2=3;
break;
case ')': str2=4;
break;
```

```
case '$': str2=5;
break;
}
if(m[str1][str2][0]=='\0')
{
printf("\nERROR");
exit(0);
}
else if(m[str1][str2][0]=='n')
i--;
else if(m[str1][str2][0]=='i')
stack[i]='i';
else
{
for(k=size[str1][str2]-1;k>=0;k--)
{
stack[i]=m[str1][str2][k];
i++;
}
i--;
}
for(k=0;k<=i;k++)
printf(" %c",stack[k]);
printf(" ");
for(k=j;k<=n;k++)
printf("%c",s[k]);
printf(" \n ");
```

```
}
printf("\n SUCCESS");
getch();
}
```

**Output:**



# PRACTICAL: 12

# Practical: 12

**Aim:** Implement a C program to implement LALR parsing.

**Program :**

```
#include<stdio.h>
#include<conio.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"}
};
struct 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','+','*',')','(','$'};
char nter[3]={'E','T','F'};
char states[12]={'a','b','c','d','e','f','g','h','m','j','k','l'};
char stack[100];
int top=-1;
char temp[10];
struct grammar
{
char left;
char right[5];
};
const struct grammar rl[6]={
{'E',"e+T"},
{'E',"T"},
{'T',"T*F"},
{'T',"F"},
{'F',"(E)"},
{'F',"i"},
};
void main()
{
char inp[80],x,p,dl[80],y,bl='a';
int i=0,j,k,l,n,m,c,len;
printf("Enter the input :");
scanf("%s",inp);
len=strlen(inp);
inp[len]='$';
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)
```

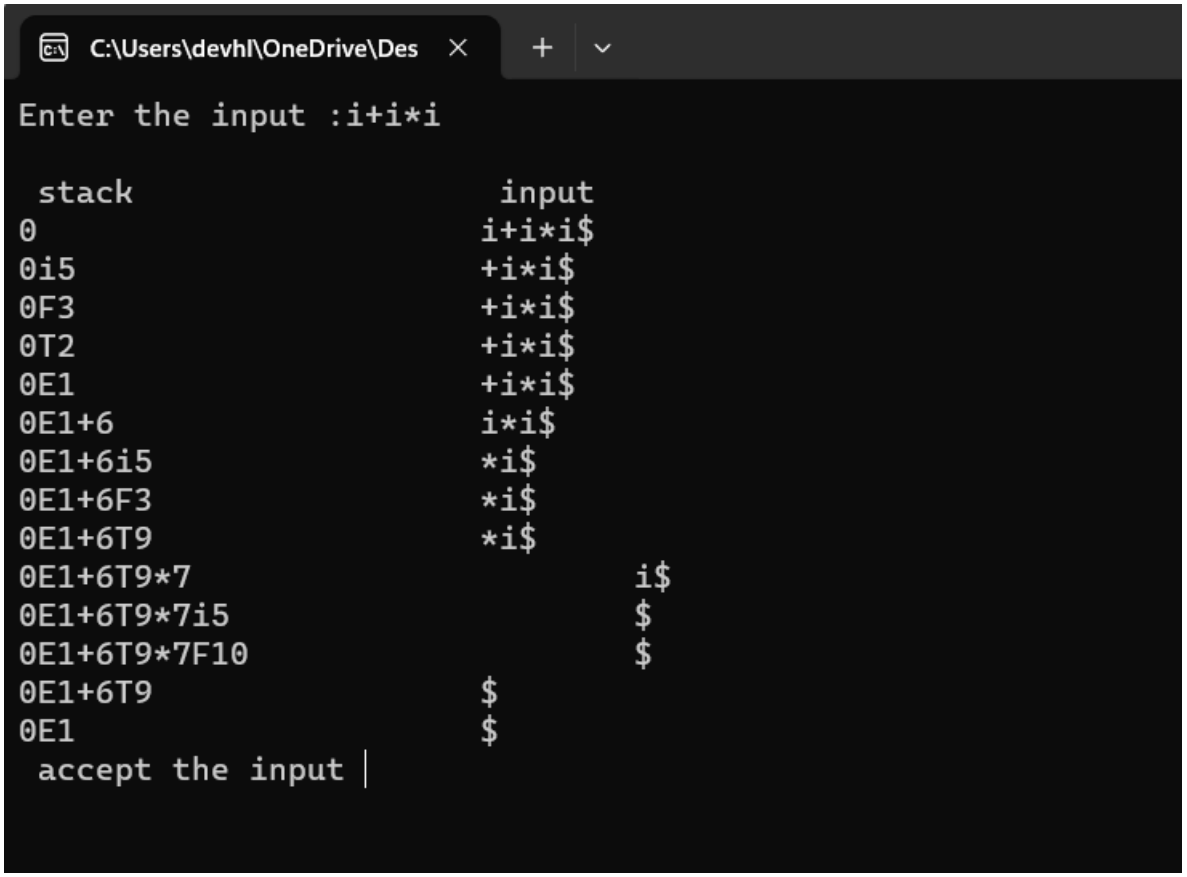
```
error();
if(strcmp(temp,"acc")==0)
break;
else
{
if(temp[0]=='s')
{
push(stack,&top,inp[i]);
push(stack,&top,temp[1]);
i++;
}
else
{
if(temp[0]=='r')
{
j=isstate(temp[1]);
strcpy(temp,rl[j-2].right);
dl[0]=rl[j-2].left;
dl[1]='\0';
n=strlen(temp);
for(k=0;k<2*n;k++)
pop(stack,&top);
for(m=0;dl[m]!='\0';m++)
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 ");
else
printf(" \n do not accept the input ");
getch();
}
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;
}
void 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);
strcpy(temp,G[k-1].r[l-1]);
}
char pop(char *s,int *sp)
{
char item;
if(*sp==-1)
printf(" stack is empty ");
else
{
item=s[*sp];
*sp=*sp-1;
}
return item;
}
void printt(char *t,int *p,char inp[],int i)
{
int r;
printf("\n");
for(r=0;r<=*p;r++)
rep(t,r);
printf("\t\t\t");
for(r=i;inp[r]!='\0';r++)
printf("%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");
break;
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:**



# PRACTICAL: 13

# Practical: 13

**Aim:** Implement a C program to implement operator precedence parsing

Program :

```
#include<stdio.h>
#include<string.h>
char *input;
int i=0;
char lasthandle[6],stack[50],handles[][5]={")E(","E*E","E+E","i","E^E"};
//(E) becomes )E( when pushed to stack
int top=0,l;
char prec[9][9]={
 /*input*/
 /*stack + - * / ^ i ( ) $ */
 /* + */ '>', '>','<','<','<','<','<','>','>',
 /* - */ '>', '>','<','<','<','<','<','>','>',
 /* * */ '>', '>','>','>','<','<','<','>','>',
 /* / */ '>', '>','>','>','<','<','<','>','>',
 /* ^ */ '>', '>','>','>','<','<','<','>','>',
/* i */ '>', '>','>','>','>','e','e','>','>',
 /* ( */ '<', '<','<','<','<','<','<','>','e',
 /* ) */ '>', '>','>','>','>','e','e','>','>',
 /* $ */ '<', '<','<','<','<','<','<','<','>',
 };
int getindex(char c)
{
switch(c)
 {
```

```
 case '+':return 0;
 case '-':return 1;
 case '*':return 2;
 case '/':return 3;
 case '^':return 4;
 case 'i':return 5;
 case '(':return 6;
 case ')':return 7;
 case '$':return 8;
 }
}
int shift()
{
stack[++top]=*(input+i++);
stack[top+1]='\0';
}
int reduce()
{
int i,len,found,t;
for(i=0;i<5;i++)//selecting handles
 {
 len=strlen(handles[i]);
 if(stack[top]==handles[i][0]&&top+1>=len)
 {
 found=1;
 for(t=0;t<len;t++)
 {
 if(stack[top-t]!=handles[i][t])
```

```
 {
 found=0;
 break;
 }
 }
 if(found==1)
 {
stack[top-t+1]='E';
 top=top-t+1;
 strcpy(lasthandle,handles[i]);
 stack[top+1]='\0';
 return 1;//successful reduction
 }
 }
 }
return 0;
}
void dispstack()
{
int j;
for(j=0;j<=top;j++)
printf("%c",stack[j]);
}
void dispinput()
{
int j;
for(j=i;j<l;j++)
 printf("%c",*(input+j));
```

```
}
void main()
{
int j;
input=(char*)malloc(50*sizeof(char));
printf("\nEnter the string\n");
scanf("%s",input);
input=strcat(input,"$");
l=strlen(input);
strcpy(stack,"$");
printf("\nSTACK\tINPUT\tACTION");
while(i<=l)
{
shift();
printf("\n");
dispstack();
printf("\t");
dispinput();
printf("\tShift");
if(prec[getindex(stack[top])][getindex(input[i])]=='>')
{
while(reduce())
{
printf("\n");
dispstack();
printf("\t");
dispinput();
printf("\tReduced: E->%s",lasthandle);
```

```
}
}
}
if(strcmp(stack,"$E$")==0)
 printf("\nAccepted;");
else
 printf("\nNot Accepted;");
}
```

**Output :**