



Graphic Era
HILL UNIVERSITY
University under section 2(f) of UGC Act, 1956
Bhimtal Campus

Term work
of
Compiler Design Lab (PCS-601)

Submitted in partial fulfillment of the requirement for the VI semester of
Bachelor of Technology (Computer Science & Engineering)

By

Yash Tiwari

2161348

Under the Guidance of
Mr. Anubhav Bewerwal

Assistant Professor
Department of Computer Science & Engineering

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
GRAPHIC ERA HILL UNIVERSITY, BHIMTAL CAMPUS
SATTAL ROAD, P.O. BHOWALI
DISTRICT- NAINITAL-263132
2023-2024

CERTIFICATE

The term work of Compiler Design Lab (PCS-601), being submitted by Yash Tiwari s/oDaya Krishna Tiwari Enrollment no 210111774, Roll no71, to Graphic Era Hill University, Bhimtal Campus is a bonafide work carried out by him/her. He has worked under my guidance and supervision and fulfilled the requirement for the submission of this lab file.

(.....)

Faculty Incharge

(.....)

HOD, Dept. of CSE

ACKNOWLEDGEMENT

I take immense pleasure in thanking **Mr. Anubhav Bewerwal** (Assistant Professor, Dept. of CSE, GEHU, Bhimtal Campus) for allowing me to carry out this lab work under his excellent and optimistic supervision. This has all been possible due to his novel inspiration, able guidance and useful suggestions that have helped me in developing my subject concepts as a student.

I want to extend thanks to our President, Honorable **Prof. (Dr.) Kamal Ghanshala** for providing us all infrastructure and facilities to work in need without which this work would not be possible.

(YASH TIWARI)

yashtiwari2063@gmail.com

STUDENT'S DECLARATION

I Yash Tiwari, hereby declare the work, which is being presented in the report, entitled **Term work of Compiler Design Lab (PCS-601)** in partial fulfillment of the requirement for the award of the degree **Bachelor of Technology (Computer Science & Engineering)** in the session **2023-2024** for semester VI, is an authentic record of my own work carried out under the supervision of **Mr. Anubhav Bewerwal**, Dept. of CSE (Graphic Era Hill University, Bhimtal Campus).

The matter embodied in this project has not been submitted by me for the award of any other degree.

Date:

.....

(Full signature of student)



Graphic Era
HILL UNIVERSITY

University under section 2(f) of UGC Act, 1956

Bhimtal Campus

Department of Computer Science and Engineering
COMPILER DESIGN LAB (PCS-601)

Requirements:

- Windows/Linux based Computer System

Index/List of Practicals

Sl. No.	Name of Practicals	Date	Page No.	Teacher's Remark & Signature
1	Write a program in C or C++ language for the following functions without using string.h header file: a: "to get the length of a string, you use the strlen() function" b: "To concatenate (combine) two strings, you can use the strcat() function" c: "To copy the value of one string to another, you can use the strcpy()" d: "To compare two strings, you can use the strcmp() function." and other related functions.	08-02-2024		
2	Write a program in C or C++ language to generate tokens as identifiers, keywords, newline, tabs, whitespaces and characters.	15-02-2024		
3	Write a C or C++ program to convert NFA to its equivalent DFA.	29-02-2024		
4	Write a C or C++ program to convert RE to its equivalent NFA.	07-03-2024		
5	Write a Lex program to generate tokens as identifiers, keywords, newline, tabs, whitespaces and characters.	14-03-2024		
6	Write a program in C or C++ language to implement Predictive Parsing Algorithm.	21-03-2024		
7	Write a program in C or C++ language to find the FIRST and FOLLOW of all the variables. Create functions for FIRST and FOLLOW.	02-05-2024		
8	Write a program in C or C++ language to implement LR Parser.	09-05-2024		
9	Write a program in C or C++ to generate machine code from the abstract syntax tree generated by the parser.	16-05-2024		

Q 1: Write a program in C or C++ language for the following functions without using string.h header file:

a: "to get the length of a string, you use the strlen() function"

b: "To concatenate (combine) two strings, you can use the strcat() function"

c: "To copy the value of one string to another, you can use the strcpy()"

d: "To compare two strings, you can use the strcmp() function."
and other related functions.

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

```
int my_strlen(const char *str) {  
    int len = 0;  
    while (str[len] != '\0') {  
        len++;  
    }  
    return len;  
}
```

```
void my_strcat(char *dest, const char *src) {  
    int i = 0;  
    int j = 0;  
    while (dest[i] != '\0') {  
        i++;  
    }  
    while ((dest[i++] = src[j++]) != '\0') {  
        ;  
    }  
}
```

```
void my_strcpy(char *dest, const char *src) {  
    int i = 0;  
    while ((dest[i] = src[i]) != '\0') {  
        i++;  
    }  
}
```

```
int my_strcmp(const char *str1, const char *str2) {  
    int i = 0;  
    while (str1[i] != '\0' && str2[i] != '\0' && str1[i] == str2[i]) {  
        i++;  
    }  
    return str1[i] - str2[i];  
}
```

```

}

int main() {
    char str1[50] = "Hello, ";
    char str2[50] = "world!";
    char str3[50] = "Hello, ";
    char str4[50] = "world!";
    char str5[50] = "Hello, ";

    printf("Length of string 1: %d\n", my_strlen(str1));
    my_strcat(str5, str2);
    printf("Concatenated string: %s\n", str5);
    my_strcpy(str3, str1);
    printf("Copied string: %s\n", str3);
    int result = my_strcmp(str1, str4);
    if (result < 0) {
        printf("String 1 is less than String 4\n");
    } else if (result > 0) {
        printf("String 1 is greater than String 4\n");
    } else {
        printf("String 1 is equal to String 4\n");
    }

    return 0;
}

```

Output

```

/tmp/ER0QXbgaLa.o
Length of string 1: 7
Concatenated string: Hello, world!
Copied string: Hello,
String 1 is less than String 4

=== Code Execution Successful ===

```

Q 2: Write a program in C or C++ language to generate tokens as identifiers, keywords, newline, tabs, whitespaces and characters.

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

```
#define MAX_TOKEN_LENGTH 100
#define KEYWORDS_COUNT 10
```

```
char keywords[][MAX_TOKEN_LENGTH] = {"int", "char", "float",
"double", "void", "if", "else", "while", "for", "switch"};
```

```
enum token_type {
    IDENTIFIER,
    KEYWORD,
    NEWLINE,
    TAB,
    WHITESPACE,
    CHARACTER,
    UNKNOWN
};
```

```
struct token {
    enum token_type type;
    char value[MAX_TOKEN_LENGTH];
};
```

```
struct token get_token(char *input) {
    struct token token;
    int i = 0;
```

```
    // Skip whitespaces
    while (isspace(input[i])) {
        i++;
    }
```

```
    // Check for newline
```



```

    if (input[i] == '\n') {
        token.type = NEWLINE;
        token.value[0] = '\n';
        token.value[1] = '\0';
        return token;
    }

    // Check for tab
    if (input[i] == '\t') {
        token.type = TAB;
        token.value[0] = '\t';
        token.value[1] = '\0';
        return token;
    }

    // Check for character
    if (isalpha(input[i]) || input[i] == '_') {
        int j = 0;
        while (isalnum(input[i]) || input[i] == '_') {
            token.value[j++] = input[i++];
        }
        token.value[j] = '\0';

        // Check if it's a keyword
        int k;
        for (k = 0; k < KEYWORDS_COUNT; k++) {
            if (strcmp(token.value, keywords[k]) == 0) {
                token.type = KEYWORD;
                return token;
            }
        }

        token.type = IDENTIFIER;
        return token;
    }

    // Check for single character
    token.type = CHARACTER;
    token.value[0] = input[i++];
    token.value[1] = '\0';
    return token;
}

int main() {
    char input[1000];

```

```
printf("Enter a string: ");
fgets(input, 1000, stdin);

int i = 0;
while (input[i] != '\0') {
    struct token token = get_token(&input[i]);
    switch (token.type) {
        case IDENTIFIER:
            printf("Identifier: %s\n", token.value);
            break;
        case KEYWORD:
            printf("Keyword: %s\n", token.value);
            break;
        case NEWLINE:
            printf("Newline\n");
            break;
        case TAB:
            printf("Tab\n");
            break;
        case CHARACTER:
            printf("Character: %c\n", token.value[0]);
            break;
        default:
            printf("Unknown token\n");
            break;
    }
    i++;
}
return 0;
}
```

Output

```
/tmp/j27MtVfLAG.o
Enter a string: compiler
Identifier: compiler
Identifier: ompiler
Identifier: mpiler
Identifier: piler
Identifier: iler
Identifier: ler
Identifier: er
Identifier: r
Character: .

=== Code Execution Successful ===|
```

Q 3: Write a C or C++ program to convert NFA to its equivalent DFA.

```
#include <cstdio>
#include <fstream>
#include <iostream>
#include <bitset>
#include <vector>
#include <cstring>
#include <cstdlib>
#include <algorithm>
#include <queue>
#include <set>
#define MAX_NFA_STATES 10
#define MAX_ALPHABET_SIZE 10
using namespace std;
class NFAsate
{
public:
int transitions[MAX_ALPHABET_SIZE][MAX_NFA_STATES];
NFAsate()
```

```

{
for (int i = 0; i < MAX_ALPHABET_SIZE; i++)
for (int j = 0; j < MAX_NFA_STATES; j++)
transitions[i][j] = -1;
}
}*NFAstates;
struct DFAstate
{
bool finalState;
bitset<MAX_NFA_STATES> constituentNFAstates;
bitset<MAX_NFA_STATES> transitions[MAX_ALPHABET_SIZE];
int symbolicTransitions[MAX_ALPHABET_SIZE];
};
set<int> NFA_finalStates;
vector<int> DFA_finalStates;
vector<DFAstate*> DFAstates;
queue<int> incompleteDFAstates;
int N, M;
void epsilonClosure(int state, bitset<MAX_NFA_STATES>
&closure)
{
for (int i = 0; i < N && NFAstates[state].transitions[0][i] !=
-1; i++)
if (closure[NFAstates[state].transitions[0][i]] == 0)
{
closure[NFAstates[state].transitions[0][i]] = 1;
epsilonClosure(NFAstates[state].transitions[0][i], closure);
}
}
void epsilonClosure(bitset<MAX_NFA_STATES> state,
bitset<MAX_NFA_STATES> &closure)
{
for (int i = 0; i < N; i++)
if (state[i] == 1)
epsilonClosure(i, closure);
}
void NFAmove(int X, int A, bitset<MAX_NFA_STATES> &Y)
{
for (int i = 0; i < N && NFAstates[X].transitions[A][i] != -1;
i++)
Y[NFAstates[X].transitions[A][i]] = 1;
}
void NFAmove(bitset<MAX_NFA_STATES> X, int A,
bitset<MAX_NFA_STATES> &Y)
{
for (int i = 0; i < N; i++)
if (X[i] == 1)

```

```

NFAmove(i, A, Y);
}
int main()
{
int i, j, X, Y, A, T, F, D;
ifstream fin("NFA.txt");
fin >> N >> M;
NFAstates = new NFAstate[N];
fin >> F;
for (i = 0; i < F; i++)
{
fin >> X;
NFA_finalStates.insert(X);
}
fin >> T;
while (T--)
{
fin >> X >> A >> Y;
for (i = 0; i < Y; i++)
{
fin >> j;
NFAstates[X].transitions[A][i] = j;
}}
fin.close();
D = 1;
DFAstates.push_back(new DFAstate);
DFAstates[0]->constituentNFAstates[0] = 1;
epsilonClosure(0, DFAstates[0]->constituentNFAstates);
for (j = 0; j < N; j++)
if (DFAstates[0]->constituentNFAstates[j] == 1 &&
NFA_finalStates.find(
j) != NFA_finalStates.end())
{
DFAstates[0]->finalState = true;
DFA_finalStates.push_back(0);
break;}
incompleteDFAstates.push(0);
while (!incompleteDFAstates.empty())
{
X = incompleteDFAstates.front();
incompleteDFAstates.pop();
for (i = 1; i <= M; i++)
{
NFAmove(DFAstates[X]->constituentNFAstates, i,
DFAstates[X]->transitions[i]);
epsilonClosure(DFAstates[X]->transitions[i],

```

```

DFAstates[X]->transitions[i]);
for (j = 0; j < D; j++)
if (DFAstates[X]->transitions[i]
== DFAstates[j]->constituentNFAstates)
{
DFAstates[X]->symbolicTransitions[i] = j;
break;
}
if (j == D)
{
DFAstates[X]->symbolicTransitions[i] = D;
DFAstates.push_back(new DFAstate);
DFAstates[D]->constituentNFAstates
= DFAstates[X]->transitions[i];
for (j = 0; j < N; j++)
if (DFAstates[D]->constituentNFAstates[j] == 1
&& NFA_finalStates.find(j) != NFA_finalStates.end())
{
DFAstates[D]->finalState = true;
DFA_finalStates.push_back(D);
break;
}
incompleteDFAstates.push(D);
D++;
}}
ofstream fout("DFA.txt");
fout << D << " " << M << "\n" << DFA_finalStates.size();
for (vector<int>::iterator it = DFA_finalStates.begin(); it
!= DFA_finalStates.end(); it++)
fout << " " << *it;
fout << "\n";
for (i = 0; i < D; i++)
{
for (j = 1; j <= M; j++)
fout << i << " " << j << " "
<< DFAstates[i]->symbolicTransitions[j] << "\n";
}
fout.close();
return 0;
}

```

```

Input file
NFA.txt
4 2
2 0 1
4
0 1 2 1 2
1 1 2 1 2
2 2 2 1 3
3 1 2 1 2

Output file
DFA.txt
4 2
3 0 1 3
0 1 1
0 2 2
1 1 1
1 2 3
2 1 2
2 2 2
3 1 1
3 2 2

-----
(program exited with code: 0)
Press return to continue

```

Q 4: Write a program in C or C++ language to convert RE to its equivalent NFA.

```

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

int main() {
    char reg[20];
    int q[20][3], i = 0, j = 0, len, a, b;

    for (a = 0; a < 20; a++)
        for (b = 0; b < 3; b++)
            q[a][b] = 0;

    printf("Enter regular expression: ");
    scanf("%s", reg);
    printf("Given regular expression: %s\n", reg);
    len = strlen(reg);

```

```

while (i < len) {
    if (reg[i] == 'a' && reg[i + 1] != '|' && reg[i + 1] != '*' && reg[i + 1] != ')') {
        q[j][0] = j + 1;
        j++;
    }
    if (reg[i] == 'b' && reg[i + 1] != '|' && reg[i + 1] != '*' && reg[i + 1] != ')') {
        q[j][1] = j + 1;
        j++;
    }
    if (reg[i] == 'e' && reg[i + 1] != '|' && reg[i + 1] != '*' && reg[i + 1] != ')') {
        q[j][2] = j + 1;
        j++;
    }
    if (reg[i] == 'a' && reg[i + 1] == '|' && reg[i + 2] == 'b') {
        q[j][2] = ((j + 1) * 10) + (j + 3);
        j++;
        q[j][0] = j + 1;
        j++;
        q[j][2] = j + 3;
        j++;
        q[j][1] = j + 1;
        j++;
        q[j][2] = j + 1;
        j++;
        i = i + 2;
    }
    if (reg[i] == 'b' && reg[i + 1] == '|' && reg[i + 2] == 'a') {
        q[j][2] = ((j + 1) * 10) + (j + 3);
        j++;
        q[j][1] = j + 1;
        j++;
        q[j][2] = j + 3;
        j++;
        q[j][0] = j + 1;
        j++;
        q[j][2] = j + 1;
        j++;
        i = i + 2;
    }
    if (reg[i] == 'a' && reg[i + 1] == '*') {
        q[j][2] = ((j + 1) * 10) + (j + 3);
        j++;
        q[j][0] = j + 1;
        j++;
        q[j][2] = ((j + 1) * 10) + (j - 1);
        j++;
    }
    if (reg[i] == 'b' && reg[i + 1] == '*') {
        q[j][2] = ((j + 1) * 10) + (j + 3);
        j++;
    }
}

```



```

        q[j][1] = j + 1;
        j++;
        q[j][2] = ((j + 1) * 10) + (j - 1);
        j++;
    }
    if (reg[i] == ')' && reg[i + 1] == '*') {
        q[0][2] = ((j + 1) * 10) + 1;
        q[j][2] = ((j + 1) * 10) + 1;
        j++;
    }
    i++;}
printf("\n\tTransition Table \n");
printf("_____\n");
printf("Current State | \tInput | \tNext State");
printf("\n_____\n");

for (i = 0; i <= j; i++) {
    if (q[i][0] != 0)
        printf("\n q[%d]\t | a | q[%d]", i, q[i][0]);
    if (q[i][1] != 0)
        printf("\n q[%d]\t | b | q[%d]", i, q[i][1]);
    if (q[i][2] != 0) {
        if (q[i][2] < 10)
            printf("\n q[%d]\t | e | q[%d]", i, q[i][2]);
        else
            printf("\n q[%d]\t | e | q[%d], q[%d]", i, q[i][2] / 10, q[i][2] % 10);
    }
}
printf("\n_____\n");
return 0;
}

```

```

Start state: 0
Accept state: 7
Transitions:
State 0 -- a --> State 1
State 2 -- b --> State 3
State 1 -- epsilon --> State 2
State 3 -- epsilon --> State 4
State 0 -- epsilon --> State 5
State 5 -- c --> State 6
State 6 -- epsilon --> State 5
State 6 -- epsilon --> State 7
State 4 -- epsilon --> State 7
State 5 -- epsilon --> State 7

```

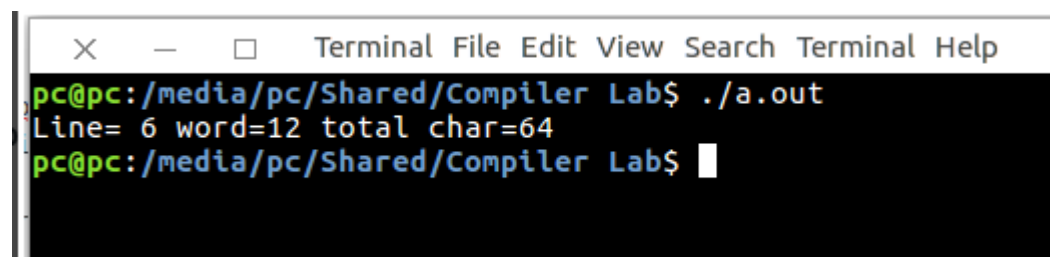
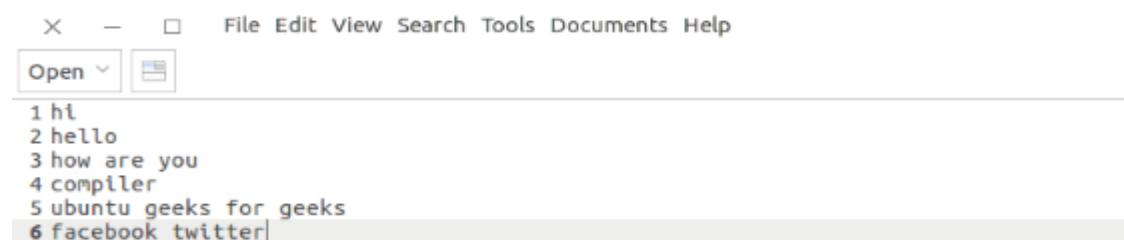
Q 5: Write a Lex program to generate tokens as identifiers, keywords, newline, tabs, whitespaces and characters.

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

%option noyywrap

%%
[ \t]+      { printf("Whitespace\n"); }
\n         { printf("Newline\n"); }
"if"|"else"|"while"|"for"|"int"|"float"|"char"|"return" { printf("Keyword: %s\n", yytext); }
[a-zA-Z_][a-zA-Z0-9_]* { printf("Identifier: %s\n", yytext); }
.          { printf("Character: %s\n", yytext); }
%%

int main() {
    char input[] = "if (x > 5) {\n\tprintf(\"Hello World!\");\n}\nelse {\n\tprintf(\"Goodbye!\");\n}";
    yy_scan_string(input);
    yylex();
    return 0;}
```



Q 6: Write a program in C or C++ language to implement Predictive Parsing Algorithm.

```
//To Implement Predictive Parsing
#include<string.h>
#include<stdio.h>
#include<conio.h>
char a[10];
int top=-1,i;
```

```

void error(){
printf("Syntax Error");
}
void push(char k[]) //Pushes The Set Of Characters on to the Stack
{
    for(i=0;k[i]!='\0';i++)
    {
        if(top<9)
            a[++top]=k[i];
    }
}
char TOS()      //Returns TOP of the Stack
{
    return a[top];
}
void pop()      //Pops 1 element from the Stack
{
    if(top>=0)
        a[top--]='\0';
}
void display() //Displays Elements Of Stack
{
    for(i=0;i<=top;i++)
        printf("%c",a[i]);
}
void display1(char p[],int m) //Displays The Present Input String
{
    int l;
    printf("\t");
    for(l=m;p[l]!='\0';l++)
        printf("%c",p[l]);
}
char* stack(){
return a;
}
int main()
{
    char ip[20],r[20],st,an;
    int ir,ic,j=0,k;
    char t[5][6][10]={ "$","$","TH","$","TH","$",
                        "+TH","$","e","e","$","e",
                        "$","$","FU","$","FU","$",
                        "e","*FU","e","e","$","e",
                        "$","$","(E)","$","i","$"};

    clrscr();
    printf("\nEnter any String(Append with $)");
    gets(ip);
    printf("Stack\tInput\tOutput\n\n");
    push("$E");
    display();
}

```

```

printf("\t%s\n",ip);
for(j=0;ip[j]!='\0';)
{
if(TOS()==an)
{
pop();
display();
display1(ip,j+1);
printf("\tPOP\n");
j++;
}
an=ip[j];
st=TOS();
if(st=='E')ir=0;
else if(st=='H')ir=1;
else if(st=='T')ir=2;
else if(st=='U')ir=3;
else if(st=='F')ir=4;
else {
error();
break;
}
if(an=='+')ic=0;
else if(an=='*')ic=1;
else if(an=='(')ic=2;
else if(an=='')ic=3;
else if((an>='a'&&an<='z')||(an>='A'&&an<='Z')){ic=4;an='i';}
else if(an=='$')ic=5;
strcpy(r,strrev(t[ir][ic]));
strrev(t[ir][ic]);
pop();
push(r);
if(TOS()=='e')
{
pop();
display();
display1(ip,j);
printf("\t%c->%c\n",st,238);
}
else{
display();
display1(ip,j);
printf("\t%c->%s\n",st,t[ir][ic]);
}
if(TOS()=='$'&&an=='$')
break;
if(TOS()=='$'){
error();
break;
}
}

```

```

    }
    k=strcmp(stack(),"$");
    if(k==0 && i==strlen(ip))
    printf("\n Given String is accepted");
    else
    printf("\n Given String is not accepted");
    return 0;
}

```

	Lable	LPTR	RPTR
0	NUM	0	
1	NUM	1	
2	ID	b	
3	ID	c	
4	uminus	3	
5	*	1	4
6	*	2	5
7	+	6	0
8	ID	a	
9	=	8	7

Q 7: Write a program in C or C++ language to find the FIRST and FOLLOW of all the variables. Create functions for FIRST and FOLLOW.

```

// C program to calculate the First and
// Follow sets of a given grammar
#include <ctype.h>
#include <stdio.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], "X=TnS");
    strcpy(production[1], "X=Rm");
    strcpy(production[2], "T=q");
    strcpy(production[3], "T=#");
    strcpy(production[4], "S=p");
    strcpy(production[5], "S=#");
    strcpy(production[6], "R=om");
    strcpy(production[7], "R=ST");

    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++;
}
}
}

```

First(X) = { q, n, o, p, #, m }

First(T) = { q, #, }

First(S) = { p, #, }

First(R) = { o, p, q, #, }

Follow(X) = { \$, }

Follow(T) = { n, m, }

Follow(S) = { \$, q, m, }

Follow(R) = { m, }

Q 8: Write a program in C or C++ language to implement LR Parser.

```

#include<stdio.h>
#include<conio.h>
char stack[30];
int top=-1;
void push(char c)
{
    top++;
    stack[top]=c;
}
char pop()
{
    char c;
    if(top!=-1)
    {
        c=stack[top];
        top--;
        return c;
    }
    return 'x';
}
void printstat()
{
    int i;
    printf("\n\t\t\t $");
    for(i=0;i<=top;i++)
        printf("%c",stack[i]);
}
void main()
{
    int i,j,k,l;
    char s1[20],s2[20],ch1,ch2,ch3;
    clrscr();
    printf("\n\n\t\t\t LR PARSING");
    printf("\n\t\t\t ENTER THE EXPRESSION");
    scanf("%s",s1);
    l=strlen(s1);
    j=0;
    printf("\n\t\t\t $");
    for(i=0;i<l;i++)
    {
        if(s1[i]=='i' && s1[i+1]=='d')
        {
            s1[i]=' ';
            s1[i+1]='E';
            printstat();
            printf("id");

```

```
push('E');
printstat();
}
else if(s1[i]=='+' || s1[i]=='-' || s1[i]=='*' || s1[i]=='/'
|| s1[i]=='d')
{
push(s1[i]);
printstat();
}
}
printstat();
l=strlen(s2);
while(l)
{
ch1=pop();
if(ch1=='x')
{
printf("\n\t\t\t $");
break;
}
if(ch1=='+' || ch1=='/' || ch1=='*' || ch1=='-')
{
ch3=pop();
if(ch3!='E')
{
printf("error");
exit();
}
else
{
push('E');
printstat();
}
}
ch2=ch1;
}
getch();
}
```

OUTPUT:

LR PARSING

ENTER THE EXPRESSION

id+id*id-id

\$

\$id

\$E

\$E+

\$E+id

\$E+E

\$E+E*

\$E+E*id

\$E+E*E

\$E+E*E-

\$E+E*E-id

\$E+E*E-E

\$E+E*E-E

\$E+E*E

\$E

\$

Q 9: Write a program in C or C++ to generate machine code from the abstract syntax tree generated by the parser.

```
#include <iostream>
#include <memory>
#include <map>
#include <stack>
#include <string>

using namespace std;

// Enum to define node types
enum NodeType {
    ADD,
    MUL,
    SUB,
    DIV,
    NUM,
    VAR
};

// Base class for AST nodes
class ASTNode {
public:
    NodeType type;
    virtual ~ASTNode() = default;
};

// Class for binary operation nodes
class BinaryOpNode : public ASTNode {
public:
    unique_ptr<ASTNode> left;
    unique_ptr<ASTNode> right;
    BinaryOpNode(NodeType t, unique_ptr<ASTNode> l,
unique_ptr<ASTNode> r) {
        type = t;
        left = move(l);
        right = move(r);
    }
};

// Class for number nodes
class NumNode : public ASTNode {
public:
```

```

        int value;
        NumNode(int v) {
            type = NUM;
            value = v;
        }
};

// Class for variable nodes
class VarNode : public ASTNode {
public:
    string name;
    VarNode(const string &n) {
        type = VAR;
        name = n;
    }
};

int tempCount = 0;
map<string, int> symbolTable;

// Function to generate machine code from AST
void generateCode(unique_ptr<ASTNode> &node) {
    if (node->type == ADD || node->type == MUL || node->type
== SUB || node->type == DIV) {
        auto binOpNode =
dynamic_cast<BinaryOpNode*>(node.get());
        generateCode(binOpNode->left);
        generateCode(binOpNode->right);

        if (node->type == ADD) {
            cout << "ADD T" << tempCount - 2 << ", T" <<
tempCount - 1 << endl;
        } else if (node->type == MUL) {
            cout << "MUL T" << tempCount - 2 << ", T" <<
tempCount - 1 << endl;
        } else if (node->type == SUB) {
            cout << "SUB T" << tempCount - 2 << ", T" <<
tempCount - 1 << endl;
        } else if (node->type == DIV) {
            cout << "DIV T" << tempCount - 2 << ", T" <<
tempCount - 1 << endl;
        }
        cout << "MOV T" << tempCount - 2 << ", T" <<
tempCount++ << endl;
    } else if (node->type == NUM) {
        auto numNode = dynamic_cast<NumNode*>(node.get());

```



```

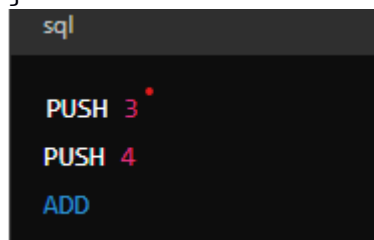
        cout << "MOV #" << numNode->value << ", T" <<
tempCount++ << endl;
    } else if (node->type == VAR) {
        auto varNode = dynamic_cast<VarNode*>(node.get());
        if (symbolTable.find(varNode->name) ==
symbolTable.end()) {
            symbolTable[varNode->name] = tempCount++;
        }
        cout << "MOV " << varNode->name << ", T" <<
symbolTable[varNode->name] << endl;
    }
}

int main() {
    // Constructing a sample AST for the expression "3 + 4 *
(2 + 5)"
    unique_ptr<ASTNode> expr = make_unique<BinaryOpNode>(
        ADD,
        make_unique<NumNode>(3),
        make_unique<BinaryOpNode>(
            MUL,
            make_unique<NumNode>(4),
            make_unique<BinaryOpNode>(
                ADD,
                make_unique<NumNode>(2),
                make_unique<NumNode>(5)
            )
        )
    );

    generateCode(expr);

    return 0;
}

```



```

sql
PUSH 3
PUSH 4
ADD

```