

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

**Harshit Lohani**

**University Roll No**

**2161167**

**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 Harshit Lohani d/o s/o…………., Enrollment no 2161167, Roll no 21011692,  
to Graphic Era Hill University, Bhimtal Campus is a bonafide work   
carried out by him. He has worked under my guidance and supervision and   
fulfilled the requirement for the submission of this lab file.**

**(…………………) (……………………)**

**Mr.Anubhav Bewerwal 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.

**(HARSHIT Lohani)**

[lohaniharshit124@gmail.com](mailto:lohaniharshit124@gmail.com)

**STUDENT’S DECLARATION**

I, **Harshit Lohani**, 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)



**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. | 17-02-2024 |  |  |
| **2** | Write a program in C or C++ language to generate tokens as identifiers, keywords, newline, tabs, whitespaces and characters. | 24-02-2024 |  |  |
| **3** | Write a C or C++ program to convert NFA to its equivalent DFA. | 06-03-2024 |  |  |
| **4** | Write a C or C++ program to convert RE to its equivalent NFA. | 13-03-2024 |  |  |
| **5** | Write a Lex program to generate tokens as identifiers, keywords, newline, tabs, whitespaces and characters. | 20-03-2024 |  |  |
| **6** | Write a program in C or C++ language to implement Predictive Parsing Algorithm. | 01-05-2024 |  |  |
| **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. | 08-05-2024 |  |  |
| **8** | Write a program in C or C++ language to implement LR Parser. | 15-05-2024 |  |  |
| **9** | Write a program in C or C++ to generate the three-address code. | 22-05-2024 |  |  |
| **10** | Write a program in C or C++ to generate machine code from the abstract syntax tree generated by the parser. | 29-05-2024 |  |  |

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>

// Function to get the length of a string

int my\_strlen(const char \*str) {

int length = 0;

while (str[length] != '\0') {

length++;

}

return length;

}

// Function to concatenate (combine) two strings

char\* my\_strcat(char \*dest, const char \*src) {

int dest\_len = my\_strlen(dest);

int i = 0;

while (src[i] != '\0') {

dest[dest\_len + i] = src[i];

i++;

}

dest[dest\_len + i] = '\0';

return dest;

}

// Function to copy the value of one string to another

char\* my\_strcpy(char \*dest, const char \*src) {

int i = 0;

while (src[i] != '\0') {

dest[i] = src[i];

i++;

}

dest[i] = '\0';

return dest;

}

// Function to compare two strings

int my\_strcmp(const char \*str1, const char \*str2) {

int i = 0;

while (str1[i] != '\0' && str2[i] != '\0') {

if (str1[i] != str2[i]) {

return str1[i] - str2[i];

}

i++;

}

return str1[i] - str2[i];

}

int main() {

char str1[100] = "Hello";

char str2[] = "World";

char str3[100];

// Test my\_strlen

printf("Length of str1: %d\n", my\_strlen(str1));

// Test my\_strcat

my\_strcat(str1, str2);

printf("Concatenated string: %s\n", str1);

// Test my\_strcpy

my\_strcpy(str3, str1);

printf("Copied string: %s\n", str3);

// Test my\_strcmp

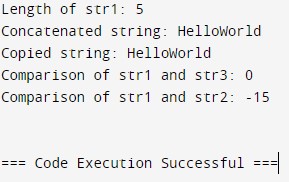
printf("Comparison of str1 and str3: %d\n", my\_strcmp(str1, str3));

printf("Comparison of str1 and str2: %d\n", my\_strcmp(str1, str2));

return 0;

}

OUTPUT:



2. 5. Write the Lex program to count the number of lines, spaces and tabs.

/\* DESCRIPTION/DEFINITION SECTION \*/

%{

#include<stdio.h>

int lc=0,sc=0,tc=0,ch=0,wc=0; // GLOBAL VARIABLES

%}

// RULE SECTION

%%

[\n] { lc++; ch+=yyleng;}

[ \t] { sc++; ch+=yyleng;}

[^\t] { tc++; ch+=yyleng;}

[^\t\n ]+ { wc++; ch+=yyleng;}

%%

int yywrap(){ return 1; }

/\* After inputting press ctrl+d \*/

// MAIN FUNCTION

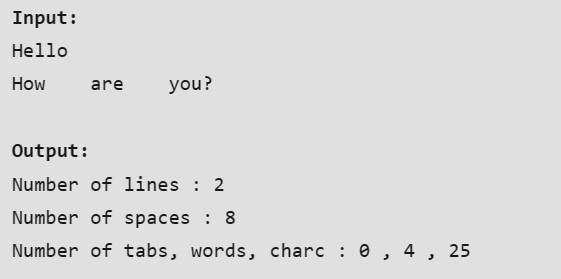
int main(){ printf("Enter the Sentence : "); yylex(); printf("Number of lines : %d\n",lc); printf("Number of spaces : %d\n",sc);

printf("Number of tabs, words, charc : %d , %d , %d\n",tc,wc,ch);

return 0;

}

OUTPUT:



3. Write a program in C or C++ language to convert NFA to its equivalent DFA.

#include<iostream>

#include<bits/stdc++.h> using namespace std;

void print(vector<vector<vector<int> > > table){ cout<<" STATE/INPUT |";

char a='a';

for(int i=0;i<table[0].size()-1;i++){ cout<<" "<<a++<<" |";

}

cout<<" ^ "<<endl<<endl;

for(int i=0;i<table.size();i++){

cout<<" "<<i<<" ";

for(int j=0;j<table[i].size();j++){

cout<<" | ";

for(int k=0;k<table[i][j].size();k++){ cout<<table[i][j][k]<<" ";

} } cout<<endl;

}

}

void printdfa(vector<vector<int> > states, vector<vector<vector<int> > > dfa){ cout<<" STATE/INPUT ";

char a='a';

for(int i=0;i<dfa[0].size();i++){ cout<<"| "<<a++<<" ";

} cout<<endl;

for(int i=0;i<states.size();i++){ cout<<"{ ";

for(int h=0;h<states[i].size();h++)

cout<<states[i][h]<<" "; if(states[i].empty()){ cout<<"^ ";

} cout<<"} ";

for(int j=0;j<dfa[i].size();j++){

cout<<" | ";

for(int k=0;k<dfa[i][j].size();k++){ cout<<dfa[i][j][k]<<" ";

}

if(dfa[i][j].empty()){ cout<<"^ ";

} } cout<<endl;

}

}

vector<int> closure(int s,vector<vector<vector<int> > > v){ vector<int> t; queue<int> q;

t.push\_back(s); int a=v[s][v[s].size()-1].size(); for(int i=0;i<a;i++){

t.push\_back(v[s][v[s].size()-1][i]); //cout<<"t[i]"<<t[i]<<endl; q.push(t[i]);

}

while(!q.empty()){ int f=q.front(); q.pop();

if(!v[f][v[f].size()-1].empty()){ int u=v[f][v[f].size()-1].size(); for(int i=0;i<u;i++){ int y=v[f][v[f].size()-1][i];

if(find(t.begin(),t.end(),y)==t.end()){ //cout<<"y"<<y<<endl; t.push\_back(y);

q.push(y);

}

}

} } return t;

} int main(){ int n,alpha;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* NFA to DFA \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl<<endl; cout<<"Enter total number of states in NFA : "; cin>>n;

cout<<"Enter number of elements in alphabet : "; cin>>alpha;

vector<vector<vector<int> > > table; for(int i=0;i<n;i++){ cout<<"For state "<<i<<endl; vector< vector< int > > v;

char a='a'; int y,yn;

for(int j=0;j<alpha;j++){ vector<int> t;

cout<<"Enter no. of output states for input "<<a++<<" : "; cin>>yn;

cout<<"Enter output states :"<<endl; for(int k=0;k<yn;k++){ cin>>y;

t.push\_back(y);

}

v.push\_back(t); } vector<int> t;

cout<<"Enter no. of output states for input ^ : "; cin>>yn;

cout<<"Enter output states :"<<endl; for(int k=0;k<yn;k++){ cin>>y;

t.push\_back(y);

}

v.push\_back(t); table.push\_back(v);

}

cout<<"\*\*\*\*\* TRANSITION TABLE OF NFA \*\*\*\*\*"<<endl; print(table);

cout<<endl<<"\*\*\*\*\* TRANSITION TABLE OF DFA \*\*\*\*\*"<<endl; vector<vector<vector<int> > > dfa; vector<vector<int> > states; states.push\_back(closure(0,table)); queue<vector<int> > q; q.push(states[0]);

while(!q.empty()){ vector<int> f=q.front(); q.pop(); vector<vector<int> > v; for(int i=0;i<alpha;i++){

vector<int> t; set<int> s;

for(int j=0;j<f.size();j++){

for(int k=0;k<table[f[j]][i].size();k++){ vector<int> cl= closure(table[f[j]][i][k],table); for(int h=0;h<cl.size();h++){ if(s.find(cl[h])==s.end()) s.insert(cl[h]);

}

}

}

for(set<int >::iterator u=s.begin(); u!=s.end();u++)

t.push\_back(\*u);

v.push\_back(t); if(find(states.begin(),states.end(),t)==states.end())

{ states.push\_back(t); q.push(t);

} }

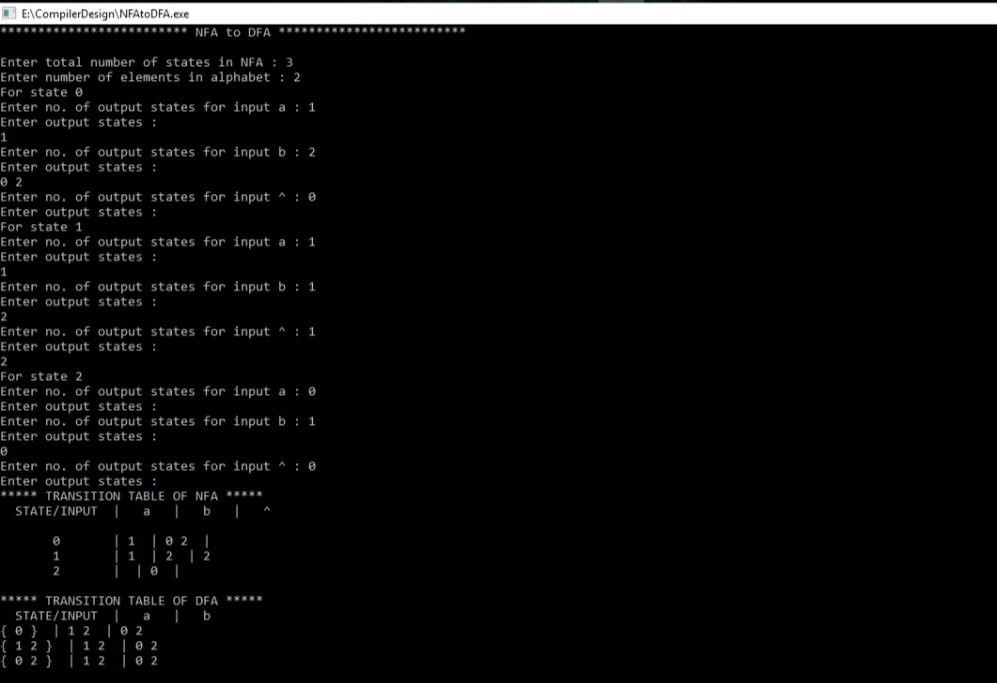
dfa.push\_back(v);

}

printdfa(states,dfa);

}

OUTPUT:



4. Write a program in C or C++ language to convert RE to its equivalent DFA.

#include<stdio.h> #include<string.h> int main()

{ char reg[20]; int q[20][3],i=0,j=1,len,a,b; for(a=0;a<20;a++) for(b=0;b<3;b++) q[a][b]=0; 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]!='\*') { q[j][0]=j+1; j++; } if(reg[i]=='b'&&reg[i+1]!='|'&&reg[i+1]!='\*') { q[j][1]=j+1; j++; } if(reg[i]=='e'&&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);

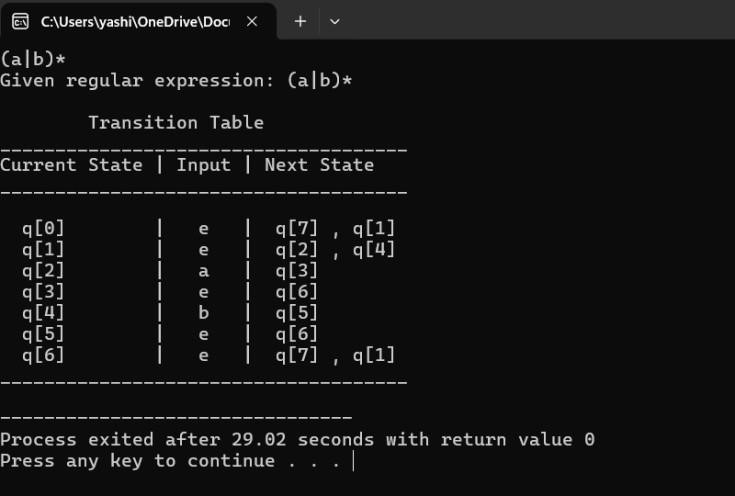
}

}

printf("\n \n"); return 0;

}

OUTPUT:



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

#include<stdio.h>

#include<conio.h>

#include<string.h>

char prol[7][10]={"s","A","A","B","B","C","C"};

char pror[7][10]={"Aa","Bb","Cd","aB","@","Cc","@"};

char prod[7][10]={"s-->A","A-->Bb","A-->Cd","B-->aB","B-->@","C-->Cc","C-->@"};

char first[7][10]={"abcd","ab",cd","a@","@","c@","@"};

char follow[7][10]={"$","$","$","a$","b$","c$","d$"};

char table[5][6][10];

{

switch(c)

{

case 'S':return0;

case 'A':return1;

case 'B':return2;

case 'C':return3;

case 'a':return0;

case 'b':return1;

case 'c':return2;

case 'd':return3;

case '$':return4;

}

retun(2);

}

void main()

{

int i,j,k;

clrscr();

for(i=0;i<5;i++)

for(j=0;j<6;j++)

strcpy(table[i][j]," ");

printf("\n The following is the predictive parsing table for the following grammar:\n");

for(i=0;i<7;i++)

printf("%s\n",prod[i]);

printf("\n Predictive parsing table is:\n ");

fflush(stdin);

for(i=0;i<7;i++)

{

k=strlen(first[i]);

for(j=0;j<10;j++)

if(first[i][j]!='@')

strcpy(table[numr(prol[i][0])+1][numr(first[i][j])+1],prod[i]);

}

for(i=0;i<7;i++)

{

if(strlen(pror[i])==1)

{

if(pror[i][0]=='@')

{

k=strlen(follow[i]);

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

strcpy(table[numr(prol[i][0])+1][numr(follow[i][j])+1]prod[i]);

}

}

}

strcpy(table[0][0]," ");

strcpy(table[0][1],"a");

strcpy(table[0][2],"b");

strcpy(table[0][3],"c");

strcpy(table[0][4],"d");

strcpy(table[0][5],"$");

strcpy(table[1][0],"S");

strcpy(table[2][0],"A");

strcpy(table[3][0],"B");

strcpy(table[4][0],"C");

printf("\n \n");

for(i-0;i<5;i++)

for(j=0;j<6;j++)

{

printf("%s\_10S",table[i][j]);

if(j==5)

printf("\n \n");

}

getch();

}

INPUT & OUTPUT:

The following is the predictive parsing table for the following grammar:

S->A

A->Bb

A->Cd

B->aB

B->@

C->Cc

C->@

Predictive parsing table is

…………………………………………..

a b c d $

…………………………………………..

S S->A S->A S->A S->A

…………………………………………..

A A->Bb A->Bb A->Cd A->Cd

…………………………………………..

B B->aB B->@ B->@ B->@

…………………………………………..

C C->@ C->@ C->@

…………………………………………..

9. Write a program in C or C++ to generate the three-address code.

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<string.h>

struct three{

char data[10],temp[7];

}s[30];

void main() {

char d1[7],d2[7]="t";

int i=0,j=1,len=0;

FILE \*f1,\*f2;

clrscr();

f1=fopen("sum.txt","r");

f2=fopen("out.txt","w");

while(fscanf(f1,"%s",s[len].data)!=EOF)

len++;

itoa(j,d1,7);

strcat(d2,d1);

strcpy(s[j].temp,d2);

strcpy(d1,"");

strcpy(d2,"t");

if(!strcmp(s[3].data,"+")) {

fprintf(f2,"%s=%s+%s",s[j].temp,s[i+2].data,s[i+4].data);

j++; }

else if(!strcmp(s[3].data,"-")){

fprintf(f2,"%s=%s-%s",s[j].temp,s[i+2].data,s[i+4].data); j++; }

for(i=4;i<len-2;i+=2)

{

itoa(j,d1,7);

strcat(d2,d1);

strcpy(s[j].temp,d2);

if(!strcmp(s[i+1].data,"+"))

fprintf(f2,"\n%s=%s+%s",s[j].temp,s[j-1].temp,s[i+2].data);

else if(!strcmp(s[i+1].data,"-"))

fprintf(f2,"\n%s=%s-%s",s[j].temp,s[j-1].temp,s[i+2].data);

strcpy(d1,"");

strcpy(d2,"t");

j++; }

fprintf(f2,"\n%s=%s",s[0].data,s[j-1].temp);

fclose(f1);

fclose(f2);

getch();

}

Input: sum.txt

out = in1 + in2 + in3 - in4

Output : out.txt

t1=in1+in2

t2=t1+in3

t3=t2-in4

out=t3

RESULT:

Thus a C program to generate a three address code for a given expression is

written, executed and the output is verified

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.

#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);

}

}

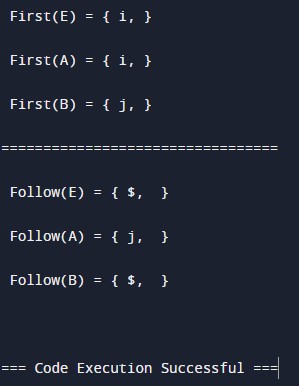
j++;

}

}

}

OUTPUT:



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 LR PARSING");

printf("\n\t\t ENTER THE EXPRESSION");

scanf("%s",s1);

l=strlen(s1);

j=0;

printf("\n\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("errror");

exit();

}

else

{

push('E');

printstat();

}

}

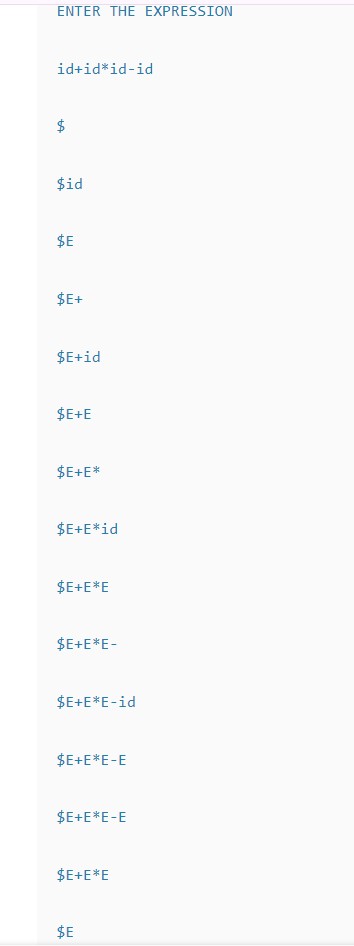
ch2=ch1;

}

getch();

}

OUTPUT:



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

#include <iostream>

#include <sstream>

#include <string>

#include <vector>

enum class NodeType {

Addition,

Multiplication,

Value

};

struct Node {

NodeType type;

int value;

Node\* left;

Node\* right;

};

Node\* createValueNode(int value) {

Node\* node = new Node;

node->type = NodeType::Value;

node->value = value;

node->left = nullptr;

node->right = nullptr;

return node;

}

Node\* createOperationNode(NodeType type, Node\* left, Node\* right) {

Node\* node = new Node;

node->type = type;

node->left = left;

node->right = right;

return node;

}

void generateAssembly(Node\* node, std::stringstream& assembly) {

if (node == nullptr) return;

switch (node->type) {

case NodeType::Addition:

generateAssembly(node->left, assembly);

generateAssembly(node->right, assembly);

assembly << "add rax, rbx\n";

break;

case NodeType::Multiplication:

generateAssembly(node->left, assembly);

generateAssembly(node->right, assembly);

assembly << "imul rax, rbx\n";

break;

case NodeType::Value:

assembly << "mov rax, " << node->value << "\n";

break;

}

}

int main() {

// Sample AST for the expression: 2 \* (3 + 4)

Node\* expr = createOperationNode(

NodeType::Multiplication,

createValueNode(2),

createOperationNode(

NodeType::Addition,

createValueNode(3),

createValueNode(4)

)

);

std::stringstream assembly;

generateAssembly(expr, assembly);

std::cout << "Generated Assembly:\n" << assembly.str() << std::endl;

// Don't forget to free memory

delete expr;

return 0;

}

OUTPUT:

