**COMPILER DESIGN LAB PROGRAMS**

**NAME:YOGESH GAUR**

**ROLL NUMBER : 1816110248**

LEXICAL ANALYZER

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include<iostream>

#include<fstream>

#include<stdlib.h>

#include<string.h>

#include<ctype.h>

using namespace std;

int isKeyword(char buffer[]){

char keywords[32][10] = {"auto","break","case","char","const","continue","default",

"do","double","else","enum","extern","float","for","goto",

"if","int","long","register","return","short","signed",

"sizeof","static","struct","switch","typedef","union",

"unsigned","void","volatile","while"};

int i, flag = 0;

for(i = 0; i < 32; ++i){

if(strcmp(keywords[i], buffer) == 0){

flag = 1;

break;

}

}

return flag;

}

int main(){

char ch, buffer[15], operators[] = "+-\*/%=";

ifstream fin("program.txt");

int i,j=0;

if(!fin.is\_open()){

cout<<"error while opening the file\n";

exit(0);

}

while(!fin.eof()){

ch = fin.get();

for(i = 0; i < 6; ++i){

if(ch == operators[i])

cout<<ch<<" is operator\n";

}

if(isalnum(ch)){

buffer[j++] = ch;

}

else if((ch == ' ' || ch == '\n') && (j != 0)){

buffer[j] = '\0';

j = 0;

if(isKeyword(buffer) == 1)

cout<<buffer<<" is keyword\n";

else

cout<<buffer<<" is indentifier\n";

}

}

fin.close();

return 0;

}

PROGRAM.TXT CONTENT

void main()

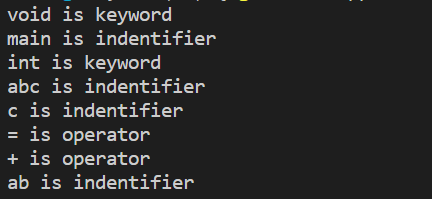
{

int a,b,c;

c = a+b;

}

OUTPUT



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

FINDING FIRST OF THE PRODUCTION

#include<stdio.h>

#include<ctype.h>

void FIRST(char );

int count,n=0;

char prodn[10][10], first[10];

main()

{

int i,choice;

char c,ch;

printf("How many productions ? :");

scanf("%d",&count);

printf("Enter %d productions epsilon= $ :\n\n",count);

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

scanf("%s%c",prodn[i],&ch);

do{

n=0;

printf("Element :");

scanf("%c",&c);

FIRST(c);

printf("\n FIRST(%c)= { ",c);

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

printf("%c ",first[i]);

printf("}\n");

printf("press 1 to continue : ");

scanf("%d%c",&choice,&ch);

}

while(choice==1);

}

void FIRST(char c)

{

int j;

if(!(isupper(c)))first[n++]=c;

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

{ if(prodn[j][0]==c)

{if(prodn[j][2]=='$')

first[n++]='$';

else

if(islower(prodn[j][2]))

first[n++]=prodn[j][2];

else

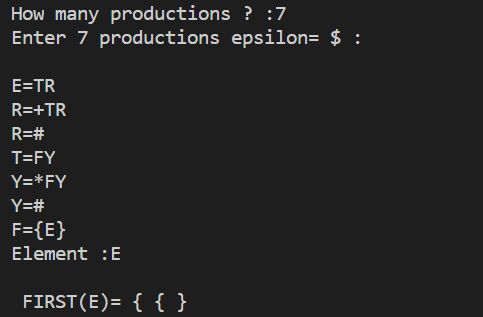
FIRST(prodn[j][2]);

}

}

}

OUTPUT



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

FOLLOW OF GRAMMAR

#include<stdio.h>

#include<string.h>

#include<ctype.h>

int n,p,i=0,j=0;

char a[10][10],Result[10];

char subResult[20];

void follow(char\* Result,char c);

void first(char\* Result,char c);

void addToResultSet(char[],char);

int main()

{

int i;

int choice;

char c,ch;

printf("Name: Vidushi Sharma\n");

printf("Roll No.: 1816110240\n");

printf("Enter the no.of productions: ");

scanf("%d", &n);

printf(" Enter %d productions\nProduction with multiple terms should be give as separate productions \n", n);

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

scanf("%s",a[i]);

do

{

printf("Find FOLLOW of -->");

scanf(" %c",&c);

follow(Result,c);

printf("FOLLOW(%c) = { ",c);

for(i=0;Result[i]!='\0';i++)

printf(" %c ",Result[i]);

printf(" }\n");

printf("Do you want to continue(Press 1 to continue)?");

scanf("%d",&choice);

}while(choice==1);

}

void follow(char\* Result,char c)

{

int k;

subResult[0]='\0';

Result[0]='\0';

if(a[0][0]==c)addToResultSet(Result,'$');

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

{

for(j=2;j<strlen(a[i]);j++)

{

if(a[i][j]==c)

{

if(a[i][j+1]!='\0')first(subResult,a[i][j+1]);

if(a[i][j+1]=='\0'&&c!=a[i][0])

follow(subResult,a[i][0]);

for(k=0;subResult[k]!='\0';k++)

addToResultSet(Result,subResult[k]);

}

}

}

}

void first(char\* R,char c)

{

int k,m;

if(!(isupper(c))&&c!='#')

addToResultSet(R,c);

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

{

if(a[k][0]==c)

{

if(a[k][2]=='#'&&c!=a[i][0])follow(R,a[i][0]);

else if((!(isupper(a[k][2])))&&a[k][2]!='#')

addToResultSet(R,a[k][2]);

else first(R,a[k][2]);

for(m=0;R[m]!='\0';m++)

addToResultSet(Result,R[m]);

}

}

}

void addToResultSet(char Result[],char val)

{

int k;

for(k=0 ;Result[k]!='\0';k++)

if(Result[k]==val)

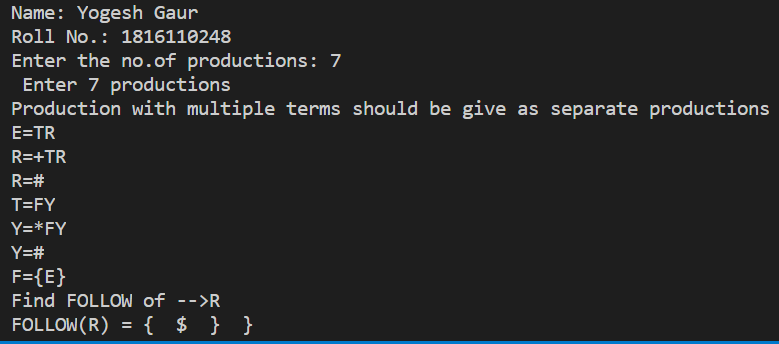
return;

Result[k]=val;

Result[k+1]='\0';

}

OUTPUT



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

INFIX TO POSTFIX

#include<iostream>

#include<stack>

#include<string>

using namespace std;

string InfixToPostfix(string expression);

int HasHigherPrecedence(char operator1, char operator2);

bool IsOperator(char C);

bool IsOperand(char C);

int main()

{

string expression;

cout<<"Enter Infix Expression \n";

getline(cin,expression);

string postfix = InfixToPostfix(expression);

cout<<"Output = "<<postfix<<"\n";

}

string InfixToPostfix(string expression)

{

stack<char> S;

string postfix = "";

for(int i = 0;i< expression.length();i++) {

if(expression[i] == ' ' || expression[i] == ',') continue;

else if(IsOperator(expression[i]))

{

while(!S.empty() && S.top() != '(' && HasHigherPrecedence(S.top(),expression[i]))

{

postfix+= S.top();

S.pop();

}

S.push(expression[i]);

}

else if(IsOperand(expression[i]))

{

postfix +=expression[i];

}

else if (expression[i] == '(')

{

S.push(expression[i]);

}

else if(expression[i] == ')')

{

while(!S.empty() && S.top() != '(') {

postfix += S.top();

S.pop();

}

S.pop();

}

}

while(!S.empty()) {

postfix += S.top();

S.pop();

}

return postfix;

}

bool IsOperand(char C)

{

if(C >= '0' && C <= '9') return true;

if(C >= 'a' && C <= 'z') return true;

if(C >= 'A' && C <= 'Z') return true;

return false;

}

bool IsOperator(char C)

{

if(C == '+' || C == '-' || C == '\*' || C == '/' || C== '$')

return true;

return false;

}

int IsRightAssociative(char op)

{

if(op == '$') return true;

return false;

}

int GetOperatorWeight(char op)

{

int weight = -1;

switch(op)

{

case '+':

case '-':

weight = 1;

case '\*':

case '/':

weight = 2;

case '$':

weight = 3;

}

return weight;

}

int HasHigherPrecedence(char op1, char op2)

{

int op1Weight = GetOperatorWeight(op1);

int op2Weight = GetOperatorWeight(op2);

if(op1Weight == op2Weight)

{

if(IsRightAssociative(op1)) return false;

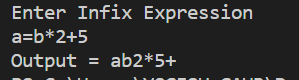
else return true;

}

return op1Weight > op2Weight ? true: false;

}

OUTPUT

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

LEAD AND TRAIL

#include <iostream>

using namespace std;

int main()

{

char s, l[20], r[10], lead[10], trail[10];

int n, j, m;

for (int i = 0; i < 10; i++)

{

lead[i] = NULL;

trail[i] = NULL;

}

cout << "\nenter total no. of productions";

cin >> n;

int k = 0;

m = 0;

for (int i = 0; i < n; i++)

{

cout << "\nenter the LHS of production";

cin >> l[i];

cout << "\nenter the RHS of production";

cin >> r;

for (int j = 0; j < 2; j++)

{

if ((r[j] == '(') || r[j] == ')' || r[j] == '\*' || r[j] == '+' || r[j] == '-' || r[j] == '/')

{

lead[k] = r[j];

k = k + 1;

}

if ((r[j] == 'i') && (r[j + 1] == 'd'))

{

lead[k] = r[j];

lead[k + 1] = r[j + 1];

k = k + 1;

}

}

for (int j = 1; j <= 2; j++)

{

if ((r[j] == '(') || r[j] == ')' || r[j] == '\*' || r[j] == '+' || r[j] == '-' || r[j] == '/')

{

trail[m] = r[j];

m = m + 1;

}

if ((r[j - 1] == 'i') && (r[j] == 'd'))

{

trail[m] = r[j - 1];

trail[m + 1] = r[j];

m = m + 1;

}

}

}

cout << "\nthe Leading(A) is :\n";

cout << "{ ";

for (int i = 0; i < k; i++)

{

if ((lead[i] == 'i') && (lead[i + 1] == 'd'))

cout << lead[i] << lead[i + 1] << " ";

else

cout << lead[i] << " ";

}

cout << "}";

cout << "\nthe Trailing(A) is :\n";

cout << "{ ";

for (int i = 0; i < m; i++)

{

if ((trail[i] == 'i') && (trail[i + 1] == 'd'))

cout << trail[i] << trail[i + 1] << " ";

else

cout << trail[i] << " ";

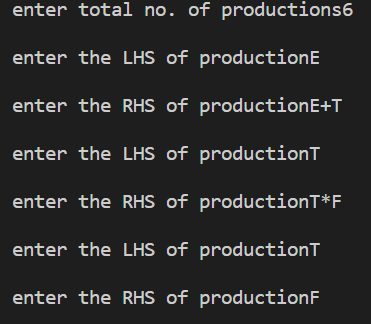
}

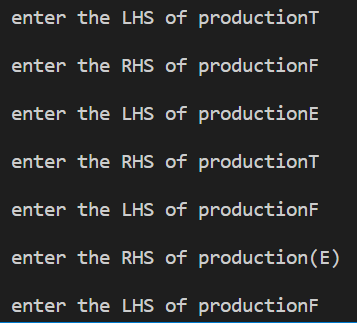
cout << "}";

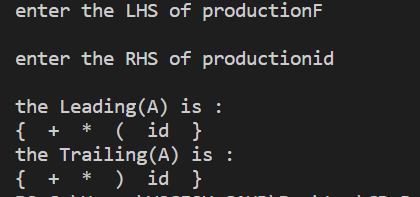
return 0;

}

OUTPUT







\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

PREFIX TO POSTFIX

#include <iostream>

#include <stack>

using namespace std;

bool isOperator(char x) {

switch (x) {

case '+':

case '-':

case '/':

case '\*':

return true;

}

return false;

}

string preToPost(string pre\_exp) {

stack<string> s;

int length = pre\_exp.size();

for (int i = length - 1; i >= 0; i--) {

if (isOperator(pre\_exp[i])) {

string op1 = s.top(); s.pop();

string op2 = s.top(); s.pop();

string temp = op1 + op2 + pre\_exp[i];

s.push(temp);

}

else {

s.push(string(1, pre\_exp[i]));

}

}

return s.top();

}

int main() {

string pre\_exp;

cout<<"Enter Prefix string : \n";

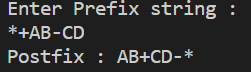
cin>>pre\_exp;

cout << "Postfix : " << preToPost(pre\_exp);

return 0;

}

OUTPUT



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

STRING ACCEPTED

#include<iostream>

#include<conio.h>

#include<string.h>

#include<stdio.h>

using namespace std;

int main()

{

char string[20];

int state=0,count=0;

cout<<"the grammar is: S->aS, S->Sb, S->ab \n";

cout<<"enter the string to be checked \n";

gets(string);

while(string[count]!='\0')

{

switch(state)

{

case 0: if (string[count]=='a')

state=1;

else

state=3;

break;

case 1: if (string[count]=='a')

state=1;

else if(string[count]=='b')

state=2;

else

state=3;

break;

case 2: if (string[count]=='b')

state=2;

else

state=3;

break;

default: break;

}

count++;

if(state==3)

break;

}

if(state==2)

cout<<"string is accepted";

else

cout<<"string is not accepted";

return 0;

}

OUTPUT

