DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ACADEMIC YEAR: 2020-21 EVEN SEMESTER

Lab Manual:

Programme (UG/PG)	:	UG
Semester	:	VI
Course Code	:	18CSC304J.
Course Title	:	COMPILER DESIGN
FACULTY	:	DR. BALAMURUGAN P

DONE BY: -

Name : PRANJAY PODDAR
Reg-No : RA1911028010129
SEC : J1

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be **University** *u/s 3 of UGC Act, 1956)*

FACULTY OF ENGINEERING AND TECHNOLOGY

SRMIST (Under section 3 of UGC Act, 1956)
SRM, Kattankulathur- 603203 Kancheepuram District

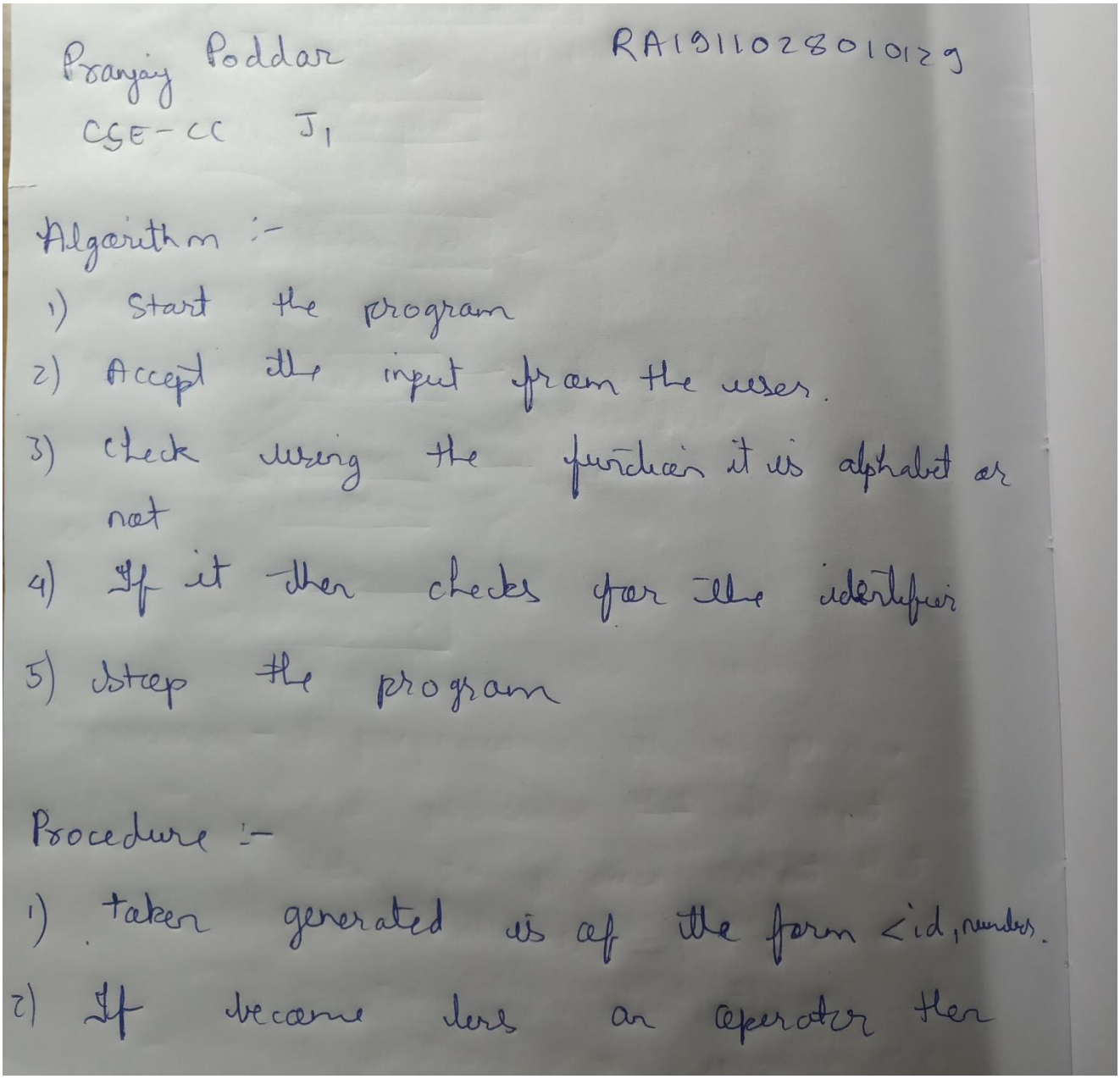
INDEX: -

Name: Pranjay Poddar Reg No: RA1911028010129 Section: J1 CSE-CC

CD EXP No. 1 (Lexical Analyser)

Aim: To implement a Lexical Analyser

Algorithm And Procedure:



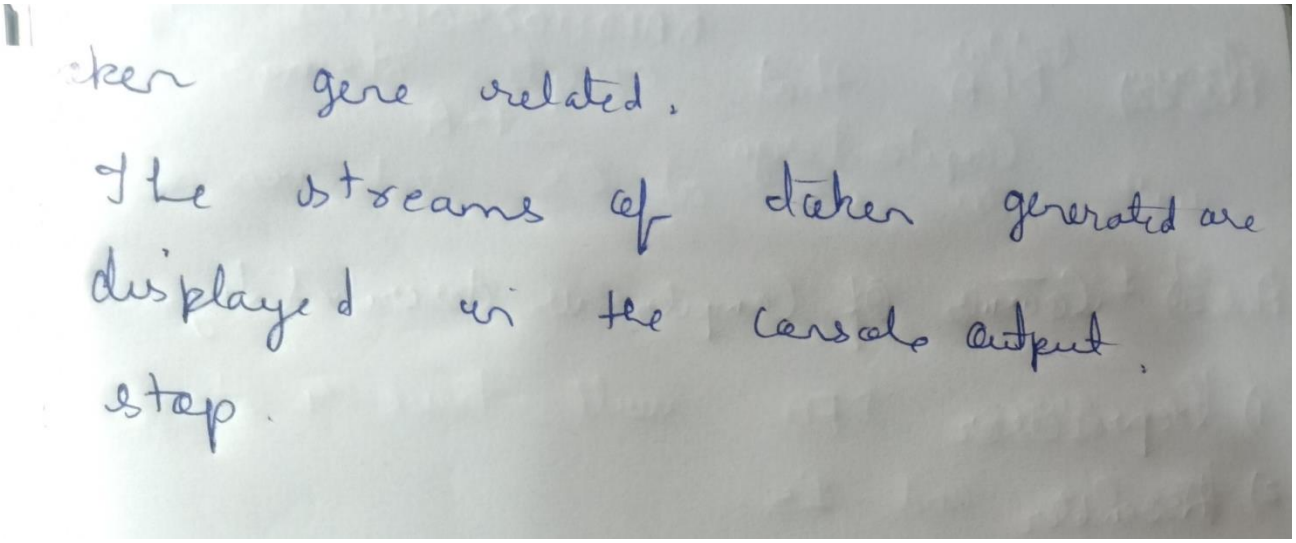
Pranjay Poddar RA1911028010129
CSE-CC J1

Algorithm :-

1) Start the program
2) Accept the input from the user.
3) Check using the function it is alphabet or not
4) If it then checks for the identifier
5) Stop the program

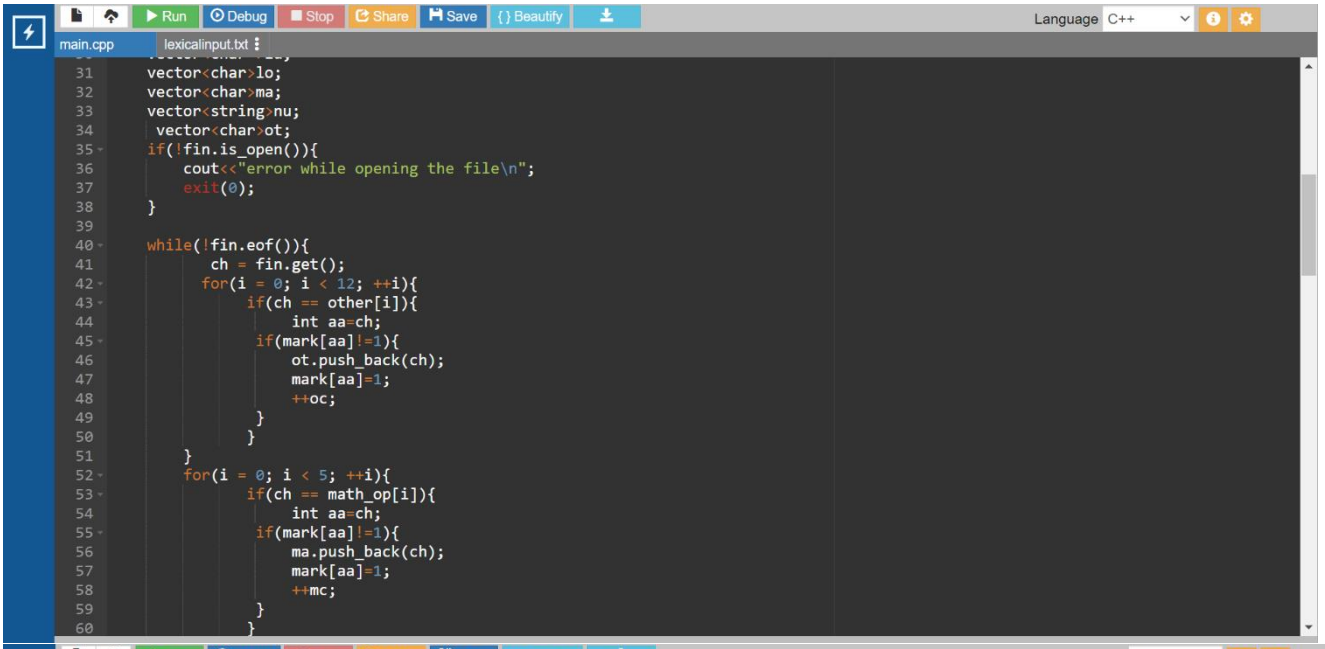
Procedure :-

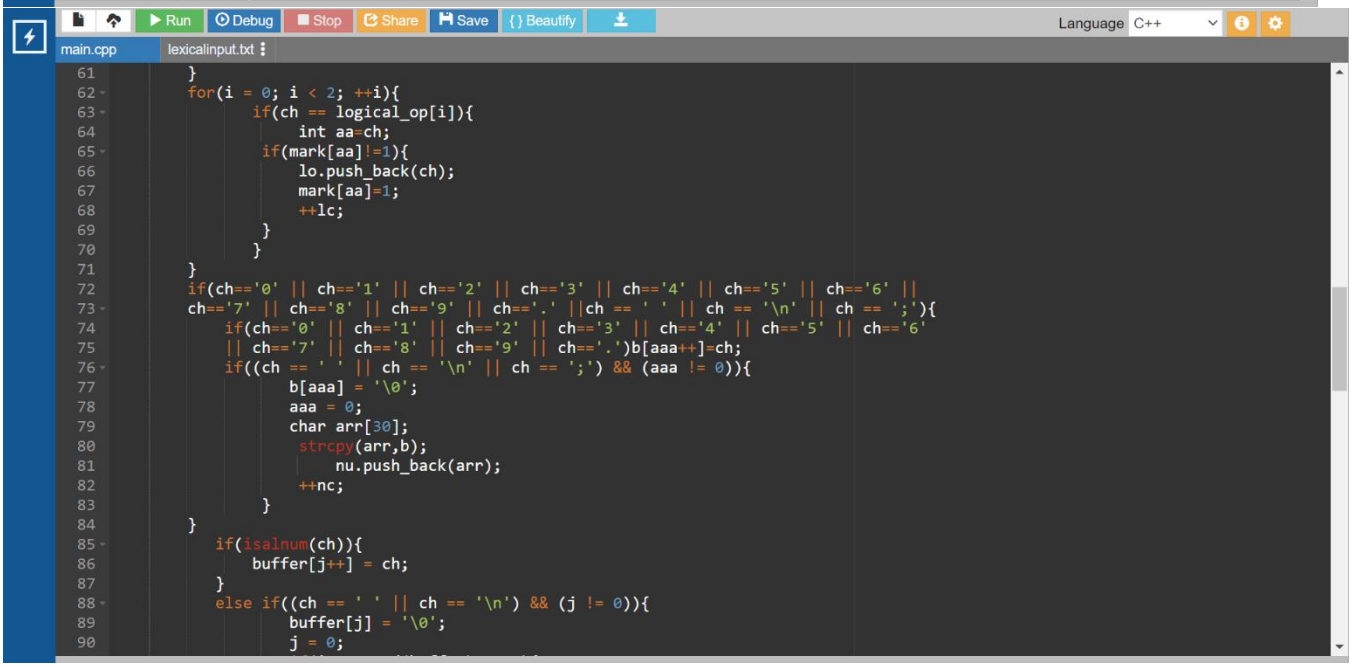
1) taken generated is of the form <id, number.
2) If became less an operator then

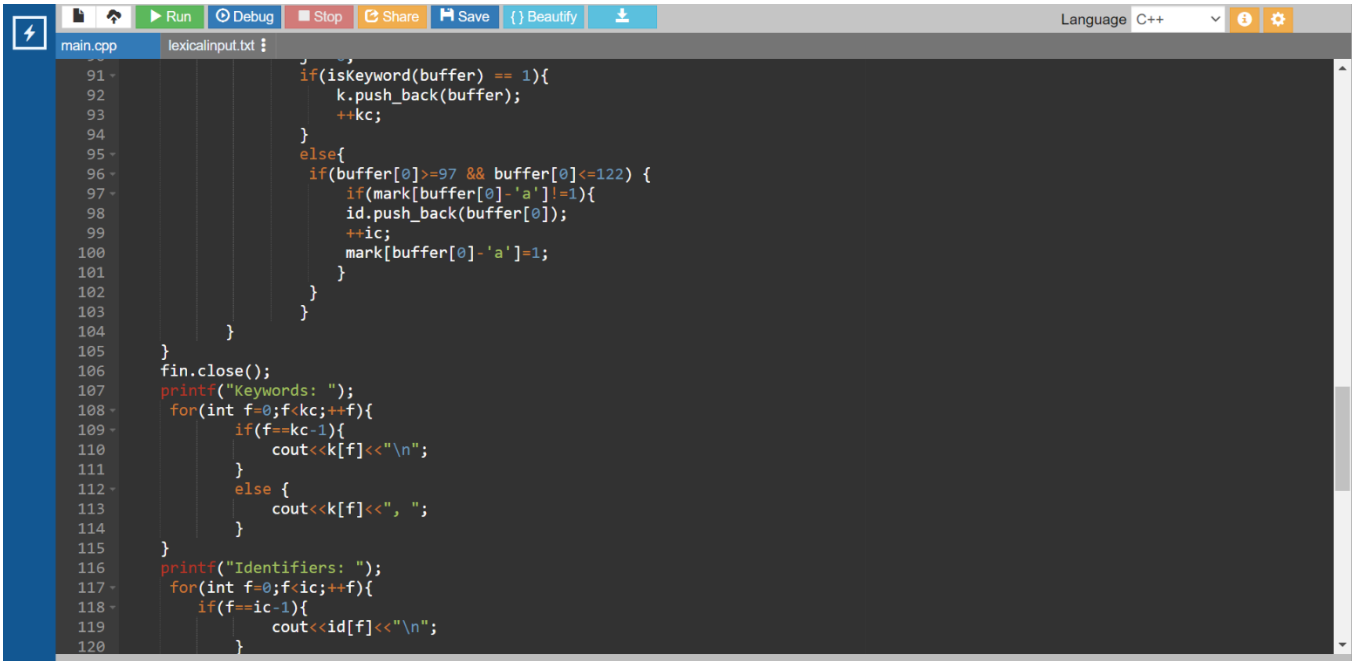


Code:

```
Run | Debug | Stop | Share | Save | {} Beautify | Language C++
main.cpp | lexicalinput.txt

#include<bits/stdc++.h>
#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],b[30], logical_op[] = "><",math_op[]="+-*/=",numer[]=".0123456789",other[]=",;\(){}[]'':";
    ifstream fin("lexicalinput.txt");
    int mark[1000]={0};
    int i,j=0,kc=0,ic=0,lc=0,mc=0,nc=0,oc=0,aaa=0;
    vector < string > k;
    vector<char >id;
```


Run
Debug
Stop
Share
Save
{ } Beautify
Language C++
main.cpp
lexicalinput.txt
vector<char>lo;
vector<char>ma;
vector<string>nu;
vector<char>ot;
if(!fin.is_open()){
cout<<"error while opening the file\n";
exit(0);
}
while(!fin.eof()){
ch = fin.get();
for(i = 0; i < 12; ++i){
if(ch == other[i]){
int aa=ch;
if(mark[aa]!=1){
ot.push_back(ch);
mark[aa]=1;
++oc;
}
}
}
for(i = 0; i < 5; ++i){
if(ch == math_op[i]){
int aa=ch;
if(mark[aa]!=1){
ma.push_back(ch);
mark[aa]=1;
++mc;
}
}


Run
Debug
Stop
Share
Save
{ } Beautify
Language C++
main.cpp
lexicalinput.txt
}
for(i = 0; i < 2; ++i){
if(ch == logical_op[i]){
int aa=ch;
if(mark[aa]!=1){
lo.push_back(ch);
mark[aa]=1;
++lc;
}
}
}
if(ch=='0' || ch=='1' || ch=='2' || ch=='3' || ch=='4' || ch=='5' || ch=='6' ||
ch=='7' || ch=='8' || ch=='9' || ch=='.' ||ch == ' ' || ch == '\n' || ch == ';'){
if(ch=='0' || ch=='1' || ch=='2' || ch=='3' || ch=='4' || ch=='5' || ch=='6'
|| ch=='7' || ch=='8' || ch=='9' || ch=='.')b[aaa++]=ch;
if((ch == ' ' || ch == '\n' || ch == ';') && (aaa != 0)){
b[aaa] = '\0';
aaa = 0;
char arr[30];
strcpy(arr,b);
nu.push_back(arr);
++nc;
}
}
if(isalnum(ch)){
buffer[j++] = ch;
}
else if((ch == ' ' || ch == '\n') && (j != 0)){
buffer[j] = '\0';
j = 0;




```
                    if(isKeyword(buffer) == 1){
                        k.push_back(buffer);
                        ++kc;
                    }
                    else{
                     if(buffer[0]>=97 && buffer[0]<=122) {
                         if(mark[buffer[0]-'a']!=1){
                         id.push_back(buffer[0]);
                         ++ic;
                         mark[buffer[0]-'a']=1;
                        }
                     }
                    }
            }
    }
    fin.close();
    printf("Keywords: ");
     for(int f=0;f<kc;++f){
            if(f==kc-1){
                cout<<k[f]<<"\n";
            }
            else {
                cout<<k[f]<<", ";
            }
    }
    printf("Identifiers: ");
     for(int f=0;f<ic;++f){
        if(f==ic-1){
                cout<<id[f]<<"\n";
            }
```


```
            else {
                cout<<id[f]<<", ";
            }
    }
    printf("Math Operators: ");
    for(int f=0;f<mc;++f){
            if(f==mc-1){
                cout<<ma[f]<<"\n";
            }
            else {
                cout<<ma[f]<<", ";
            }
    }
    printf("Logical Operators: ");
    for(int f=0;f<lc;++f){
            if(f==lc-1){
                cout<<lo[f]<<"\n";
            }
            else {
                cout<<lo[f]<<", ";
            }
    }
    printf("Numerical Values: ");
    for(int f=0;f<nc;++f){
            if(f==nc-1){
                cout<<nu[f]<<"\n";
            }
            else {
                cout<<nu[f]<<", ";
            }
```

```
    }
    printf("Logical Operators: ");
    for(int f=0;f<lc;++f){
            if(f==lc-1){
                cout<<lo[f]<<"\n";
            }
            else {
                cout<<lo[f]<<", ";
            }
    }
    printf("Numerical Values: ");
    for(int f=0;f<nc;++f){
            if(f==nc-1){
                cout<<nu[f]<<"\n";
            }
            else {
                cout<<nu[f]<<", ";
            }
    }
    printf("Others: ");
    for(int f=0;f<oc;++f){
            if(f==oc-1){
                cout<<ot[f]<<"\n";
            }
            else {
                cout<<ot[f]<<" ";
            }
    }
    return 0;
}
```

Input:

```
int z = 10;
float q = 53.5;
if(q > 10){
char name[100] = "Hello my name is Pranjay";
}
```

Output:

```
Keywords: int, float, char
Identifiers: z, q, i, n, m, p
Math Operators: =
Logical Operators: >
Numerical Values: 10, 53.5, 10, 100
Others: ; ( ) { [ ] }


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

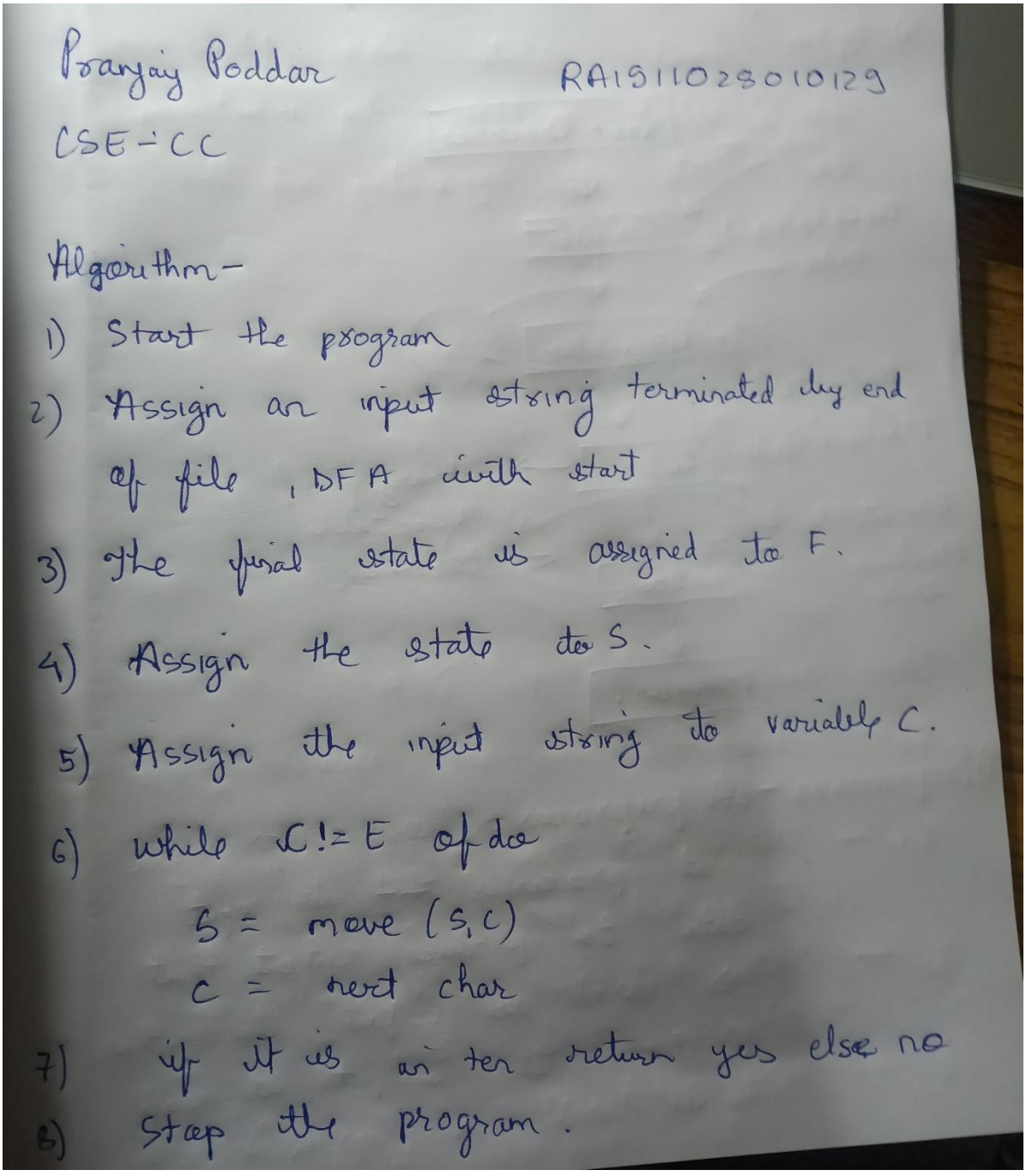
Result: Lexical Analyser is implemented successfully in online GDB compiler.

Name: Pranjay Poddar Reg No: RA1911028010129 Section: J1 CSE-CC

CD EXP No. 2 (Conversion of regular expression to NFA – Thompson's)

Aim: To implement conversion of regular expression to NFA – Thompson's

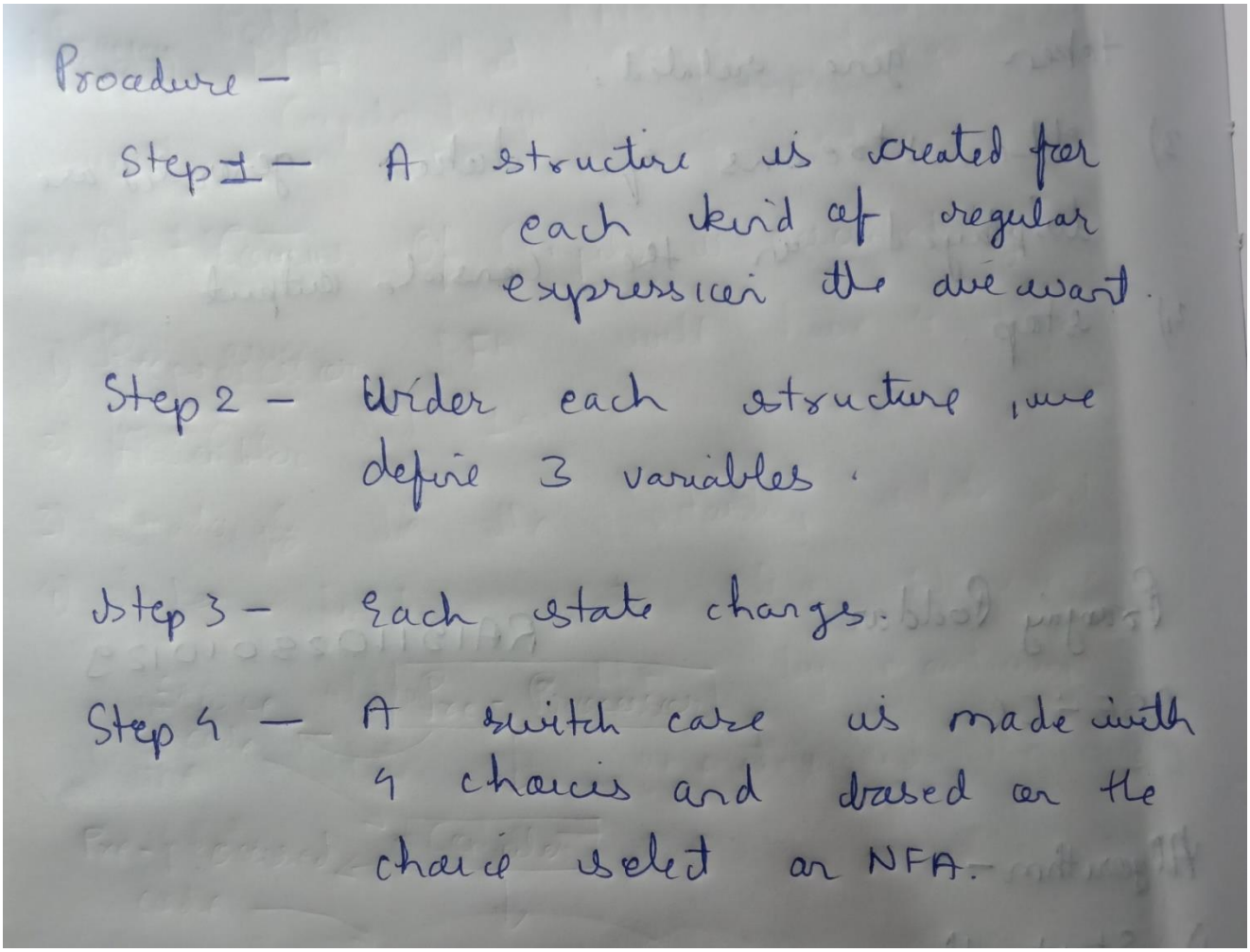
Algorithm:



Pranjay Poddar RA1911028010129
CSE-CC

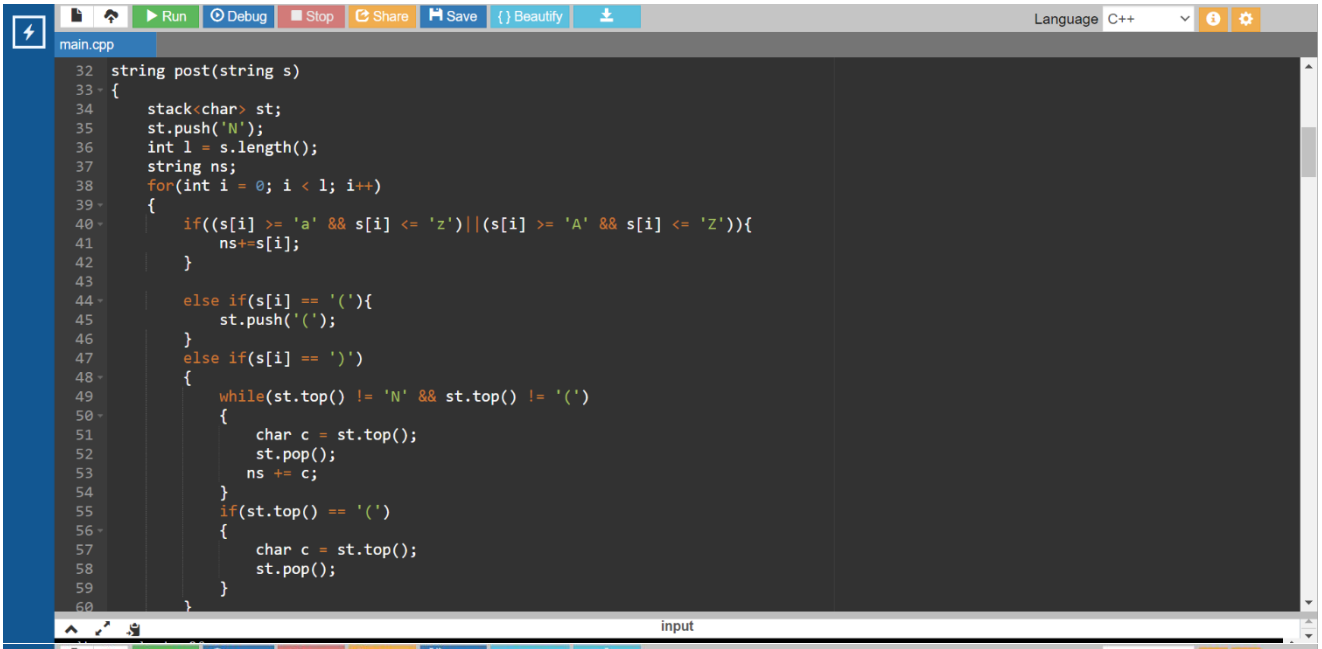
Algorithm –

1) Start the program
2) Assign an input string terminated by end of file, DFA with start
3) the final state is assigned to F.
4) Assign the state to S.
5) Assign the input string to variable C.
6) while C != E of do
   S = move (S, C)
   C = next char
7) if it is an ter return yes else no
8) Stop the program.


Procedure –
Step 1 – A structure is created for each kind of regular expression the due want.
Step 2 – Under each structure, we define 3 variables.
Step 3 – Each state changes.
Step 4 – A switch case is made with 4 choices and based on the choice select an NFA.

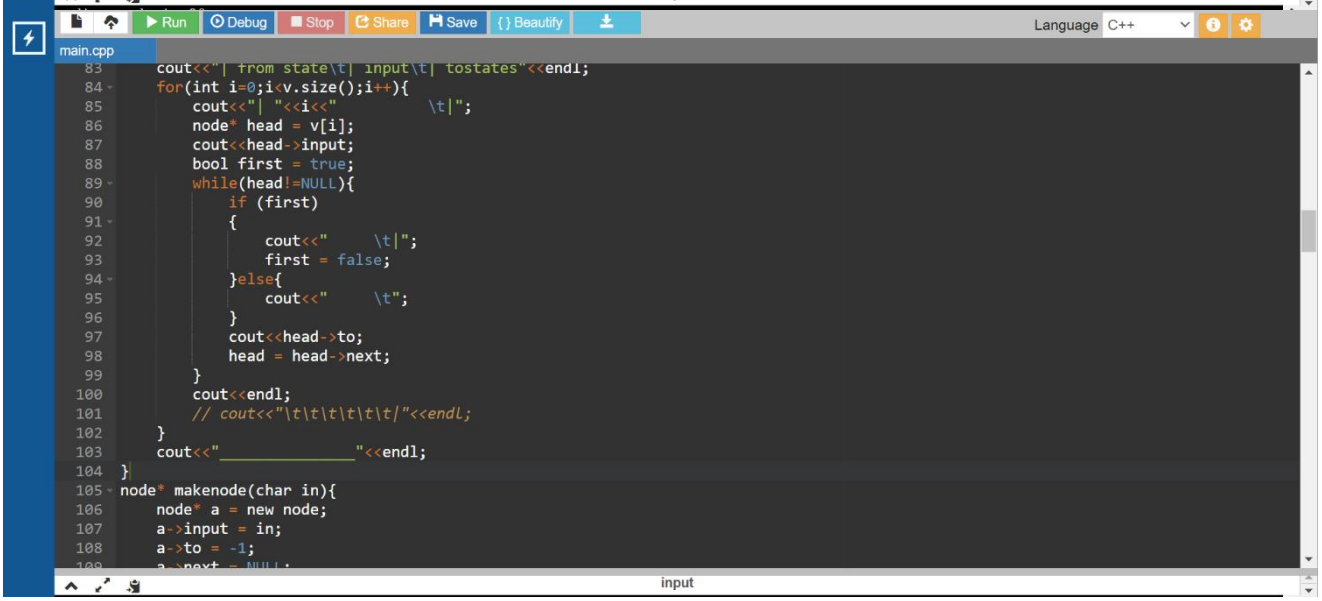
Code:


Run
Debug
Stop
Share
Save
{} Beautify
Language C++
main.cpp
#include<iostream>
#include<stack>
#include<string>
#include <algorithm>
#include<vector>
using namespace std;
class node{
public:
char input;
int to;
node *next;
};
int prec(char c){
if(c=='*'){
return 3;
}else if(c=='.'){
return 2;
}else if(c=='+'){
return 1;
}else{
return -1;
}
}
input

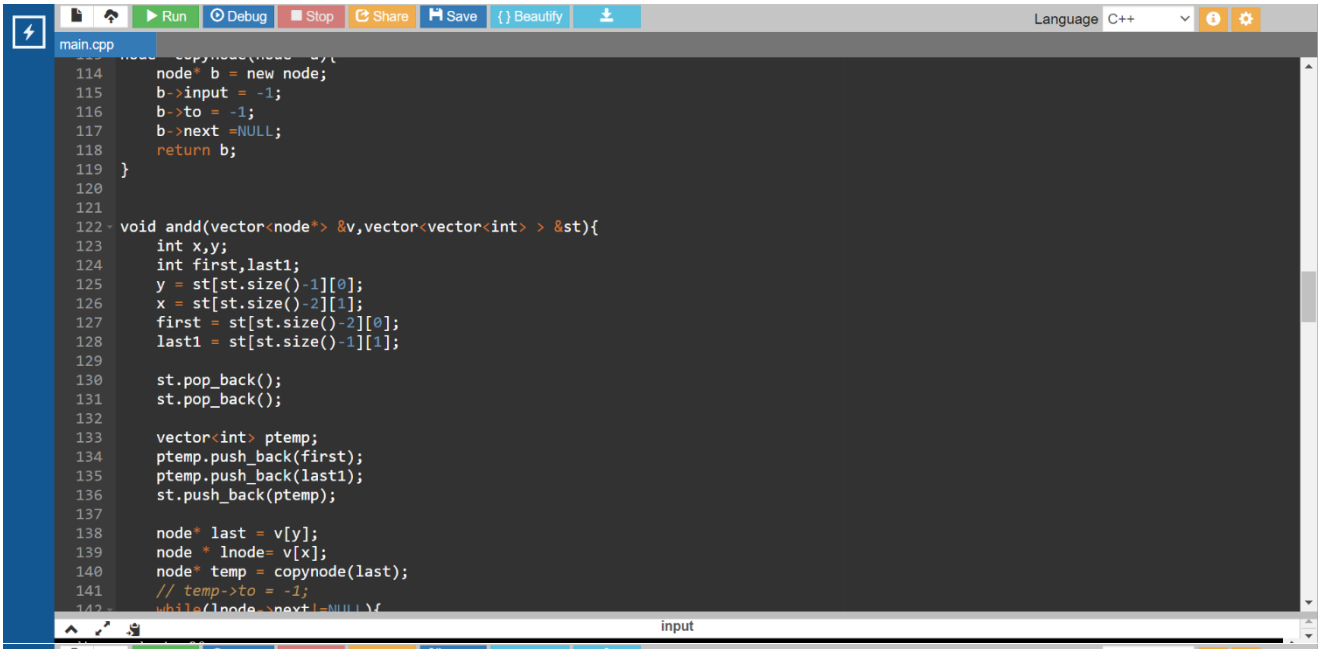



```
string post(string s)
{
    stack<char> st;
    st.push('N');
    int l = s.length();
    string ns;
    for(int i = 0; i < l; i++)
    {
        if((s[i] >= 'a' && s[i] <= 'z')||(s[i] >= 'A' && s[i] <= 'Z')){
            ns+=s[i];
        }

        else if(s[i] == '('){
            st.push('(');
        }
        else if(s[i] == ')')
        {
            while(st.top() != 'N' && st.top() != '(')
            {
                char c = st.top();
                st.pop();
               ns += c;
            }
            if(st.top() == '(')
            {
                char c = st.top();
                st.pop();
            }
        }
```


```
                char c = st.top();
                st.pop();
            }
        }
        else{
            while(st.top() != 'N' && prec(s[i]) <= prec(st.top()))
            {
                char c = st.top();
                st.pop();
                ns += c;
            }
            st.push(s[i]);
        }

    }
    while(st.top() != 'N')
    {
        char c = st.top();
        st.pop();
        ns += c;
    }
return ns;
}

void printnode(vector<node*> v){
    cout<<"_____"<<endl;
    cout<<"| from state\t| input\t| tostates"<<endl;
    for(int i=0;i<v.size();i++){
        cout<<"| "<<i<<"          \t|";
```




```
    cout<<"| from state\t| input\t| tostates"<<endl;
    for(int i=0;i<v.size();i++){
        cout<<"| "<<i<<"          \t|";
        node* head = v[i];
        cout<<head->input;
        bool first = true;
        while(head!=NULL){
            if (first)
            {
                cout<<"     \t|";
                first = false;
            }else{
                cout<<"     \t";
            }
            cout<<head->to;
            head = head->next;
        }
        cout<<endl;
        // cout<<"\t\t\t\t\t\t/"<<endl;
    }
    cout<<"_____"<<endl;
}
node* makenode(char in){
    node* a = new node;
    a->input = in;
    a->to = -1;
    a->next = NULL;
```


Run
Debug
Stop
Share
Save
{} Beautify
Language C++
main.cpp
node* b = new node;
b->input = -1;
b->to = -1;
b->next =NULL;
return b;
}
void andd(vector<node*> &v,vector<vector<int> > &st){
int x,y;
int first,last1;
y = st[st.size()-1][0];
x = st[st.size()-2][1];
first = st[st.size()-2][0];
last1 = st[st.size()-1][1];
st.pop_back();
st.pop_back();
vector<int> ptemp;
ptemp.push_back(first);
ptemp.push_back(last1);
st.push_back(ptemp);
node* last = v[y];
node * lnode= v[x];
node* temp = copynode(last);
// temp->to = -1;
input


Run
Debug
Stop
Share
Save
{} Beautify
Language C++
main.cpp
}
void orr(vector<node*> &v,vector<vector<int> > &st){
int x,y,x1,y1;
x = st[st.size()-2][0];
y = st[st.size()-1][0];
x1 = st[st.size()-2][1];
y1 = st[st.size()-1][1];
node* start = makenode('e');
node* end = makenode('e');
v.push_back(start);
int firstnode = v.size() -1;
v.push_back(end);
int endnode = v.size() -1;
st.pop_back();
st.pop_back();
vector<int> ptemp;
ptemp.push_back(firstnode);
ptemp.push_back(endnode);
st.push_back(ptemp);
for(int i=0;i<v.size()-2;i++){
node* h=v[i];
while(h->next!=NULL){
if(h->to==x || h->to == y){
h->to = firstnode;
}
input


Run
Debug
Stop
Share
Save
{} Beautify
Language C++
main.cpp
node* adlink = v[x1];
while(adlink->next!=NULL){
adlink = adlink->next;
}
adlink->to= endnode;
adlink->next = copynode(end);
node* adlink1 = v[y1];
while(adlink1->next!=NULL){
adlink1 = adlink1->next;
}
adlink1->to = endnode;
adlink1->next = copynode(end);
}
void closure(vector<node*> &v, vector<vector<int> > &st){
int x,x1;
x = st[st.size()-1][0];
x1 = st[st.size()-1][1];
node* s = makenode('e');
// node* e = makenode('e');
v.push_back(s);
int firstnode = v.size() -1;
input

```
            vector<int> ptemp;
            ptemp.push_back(v.size()-1);
            ptemp.push_back(v.size()-1);
            st.push_back(ptemp);
        }
        else if(o[1]=='.'){
            andd(v,st);
        }
        else if(o[1]=='+'){
            orr(v,st);
        }
        else if(o[1]=='*'){
            closure(v,st);
        }


    }
    cout<<"\ntrainsition table for given regular expression is - \n";
    printnode(v);
    cout<<endl;
    cout<<"starting node is ";
    cout<<st[st.size()-1][0]<<endl;
    cout<<"ending node is ";
    cout<<st[st.size()-1][1]<<endl;
    return 0;
}
```

Input:

```
Enter a regular expression
^([a-zA-Z0-9_\-\.]+)@
```

Output:

```
Enter a regular expression
^([a-zA-Z0-9_\-\.]+)@(

postfix expression is (^a[zA-Z-0-9_\-.\+](@

trainsition table for given regular expression is -
_____________
| from state    | input | tostates
| 0             |(      |-1
| 1             |^      |-1
| 2             |a      |-1
| 3             |[      |-1
| 4             |z      |-1
| 5             |A      |-1
| 6             |-      |-1
| 7             |Z      |-1
| 8             |-      |-1
| 9             |0      |-1
| 10            |-      |-1
| 11            |9      |-1
| 12            |_      |-1
| 13            |\      |14             -1
| 14            |-      |17             -1
| 15            |\      |17             -1
| 16            |e      |13             15      -1
| 17            |e      |-1
| 18            |]      |-1
| 19            |(      |-1
| 20            |@      |-1
_____________

starting node is 20
ending node is 20
```

Result: Conversion of regular expression to NFA – Thompson's is implemented successfully in online GDB compiler.

Exp No - 03
Date - 11/02/2022

Prayag Poddar
RA1911028010129
CSE - CC (J1)

Conversion of NFA to DFA

Aim :- To implement conversion of NFA to DFA

Procedure :-

Step 1 :- Get the input from the user

Step 2 :- Set the only state in SDFA to "unmarked".

Step 3 :- While SDFA contains an unmarked state do :-

a) Let T be that unmarked state

b) for each a in % do S = e-closure (move NFA(T, a)).

c) if S is not in SDFA already then, add S to SDFA (as an "unmarked" state)

d) Set Move DFA (T, a) to S.

Step 4 :- For each S in SDFA if any S & S is a final state in the NFA then, mark S as a final state in the DFA.

PROGRAM:

Run | Debug | Stop | Share | Save | {} Beautify | Language C++

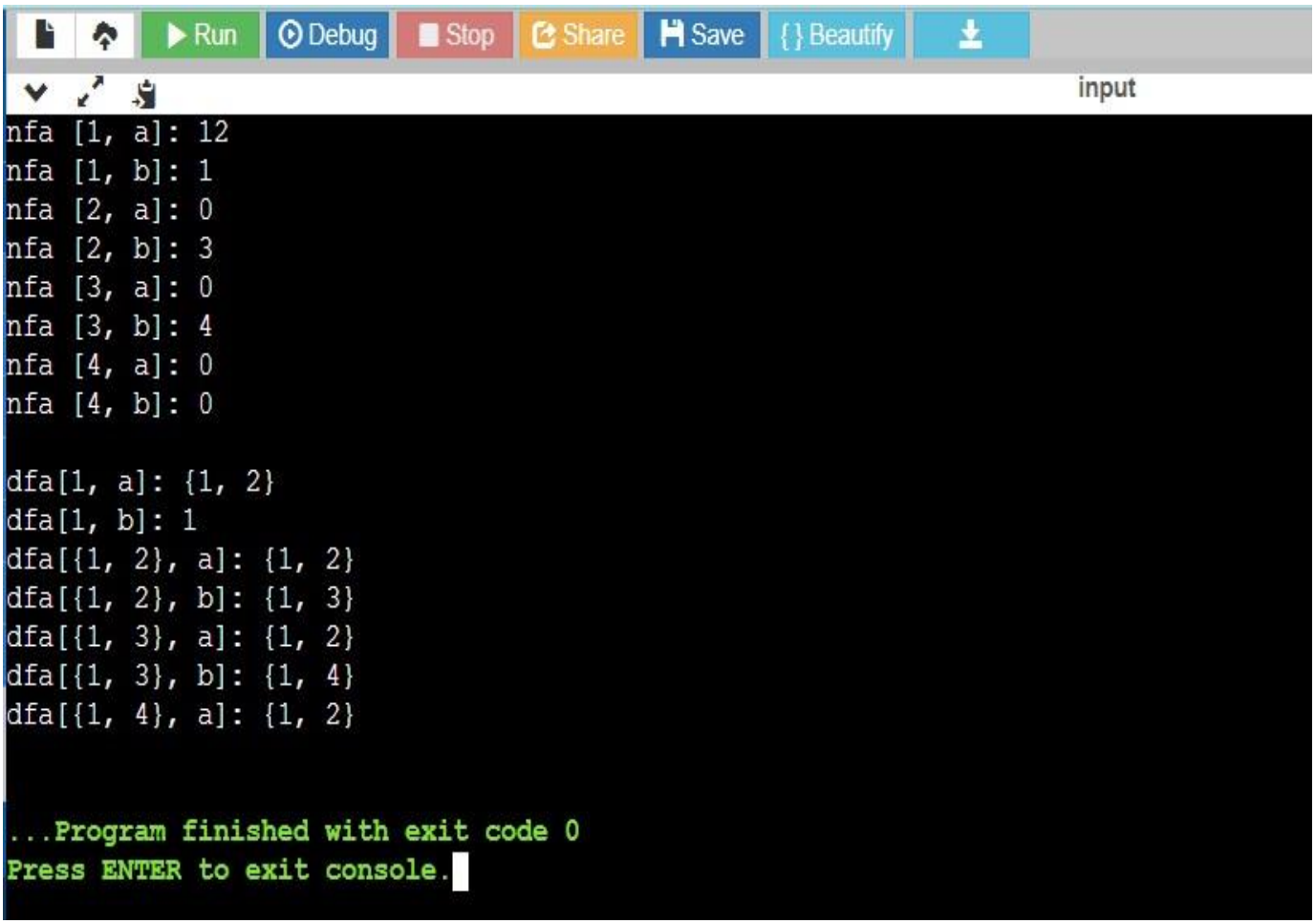
main.cpp

```cpp
#include <vector>
#include <iostream>
using namespace std;

int main()
{
    vector<vector<int>> nfa( 5 , vector<int> (3));
    vector<vector<int>> dfa( 10 , vector<int> (3));
    for(int i=1;i<5;i++){
        for(int j=1;j<=2;j++){
            int h;
            if (j == 1){
                cout << "nfa [" << i << ", a]: ";
            }
            else{
                cout << "nfa [" << i << ", b]: ";
            }
            cin>>h;
            nfa[i][j]=h;
        }
    }
    int dstate[10];
    int i=1,n,j,k,flag=0,m,q,r;
    dstate[i++]=1;
    n=i;

    dfa[1][1]=nfa[1][1];
    dfa[1][2]=nfa[1][2];
    cout<<"\n"<<"dfa["<<dstate[1]<<", a]: {"<<dfa[1][1]/10<<", "<<dfa[1][1]%10<<"}";
    cout<<"\n"<<"dfa["<<dstate[1]<<", b]: "<<dfa[1][2];

 for(j=1;j<n;j++)
    {
        if(dfa[1][1]!=dstate[j])
          flag++;
    }
        if(flag==n-1)
        {
            dstate[i++]=dfa[1][1];
            n++;
        }
        flag=0;
        for(j=1;j<n;j++)
        {
             if(dfa[1][2]!=dstate[j])
                flag++;
        }
        if(flag==n-1)
        {
             dstate[i++]=dfa[1][2];
             n++;
        }
        k=2;
        while(dstate[k]!=0)
        {
            m=dstate[k];
            if(m>10)
            {
                q=m/10;
                r=m%10;
            }
            if(nfa[r][1]!=0)
                 dfa[k][1]=nfa[q][1]*10+nfa[r][1];
            else
                dfa[k][1]=nfa[q][1];
            if(nfa[r][2]!=0)
                dfa[k][2]=nfa[q][2]*10+nfa[r][2];
            else
                dfa[k][2]=nfa[q][2];

            if (dstate[k] > 10){
                if (dfa[k][1] > 10){
```

```
            cout<<"\n"<<"dfa[{"<<dstate[k]/10 << ", " << dstate[k]%10 <<"}, a]: {"<<dfa[k][1]/10 << ", " << dfa[k][1]%10 << "}
        }
        else{
            cout<<"\n"<<"dfa[{"<<dstate[k]/10 << ", " << dstate[k]%10 <<"}, a]: "<<dfa[k][1];
        }
    }
    else{
        if (dfa[k][1] > 10){
            cout<<"\n"<<"dfa["<<dstate[k] << ", a]: {"<<dfa[k][1]/10 << ", " << dfa[k][1]%10 << "}";
        }
        else{
            cout<<"\n"<<"dfa["<<dstate[k] << ", a]: "<<dfa[k][1];
        }
    }
    if (dstate[k] > 10){
        if (dfa[k][2] > 10){
            cout<<"\n"<<"dfa[{"<<dstate[k]/10 << ", " << dstate[k]%10 <<"}, b]: {"<<dfa[k][2]/10 << ", " << dfa[k][2]%10 << "}
        }
        else{
            cout<<"\n"<<"dfa[{"<<dstate[k]/10 << ", " << dstate[k]%10 <<"}, b]: "<<dfa[k][2];
        }
    }
    else{
        if (dfa[k][1] > 10){
            cout<<"\n"<<"dfa["<<dstate[k] << ", b]: {"<<dfa[k][2]/10 << ", " << dfa[k][2]%10 << "}";
        }
        else{
            cout<<"\n"<<"dfa["<<dstate[k] << ", b]: "<<dfa[k][2];
        }
    }
   flag=0;
    for(j=1;j<n;j++)
    {
     if(dfa[k][1]!=dstate[j])
      flag++;
    }
```

```
        if(flag==n-1)
        {
            dstate[i++]=dfa[k][1];
            n++;
        }
    flag=0;
    for(j=1;j<n;j++)
    {
            if(dfa[k][2]!=dstate[j])
                flag++;
    }
    if(flag==n-1)
    {
            dstate[i++]=dfa[k][2];
            n++;
    }
    k++;
    }
    return 0;
}
```

OUTPUT:



Result : The program to convert NFA to DFA was executed successsfully in online GDB compiler.

WEEK-4 Elimination of left recursion and left factoring

Experiment - 04
17/02/22

Pranjay Poddar
RA1911028010129
CSE-CC , J1

Elimination of Left Recursion
Left Factoring

Aim:- To remove left recursive & left factoring by implementing the program.

Algorithm - (i) To remove left recursion.

→ We need to check if the grammer contains left recursion if it present we need to seperate the production & start working.

$S \rightarrow Sa/Sb/c/d$

→ Introduce a new non-terminal & write it at the rest of every terminal. We produce new non-terminal & S' & write new production as.

$S \rightarrow cS' / 2S'$

→ Newly produce non-terminal in LHS & in RHS either produce new production in which terminal & non-terminal.

$S' \rightarrow 2/aS'/b'$

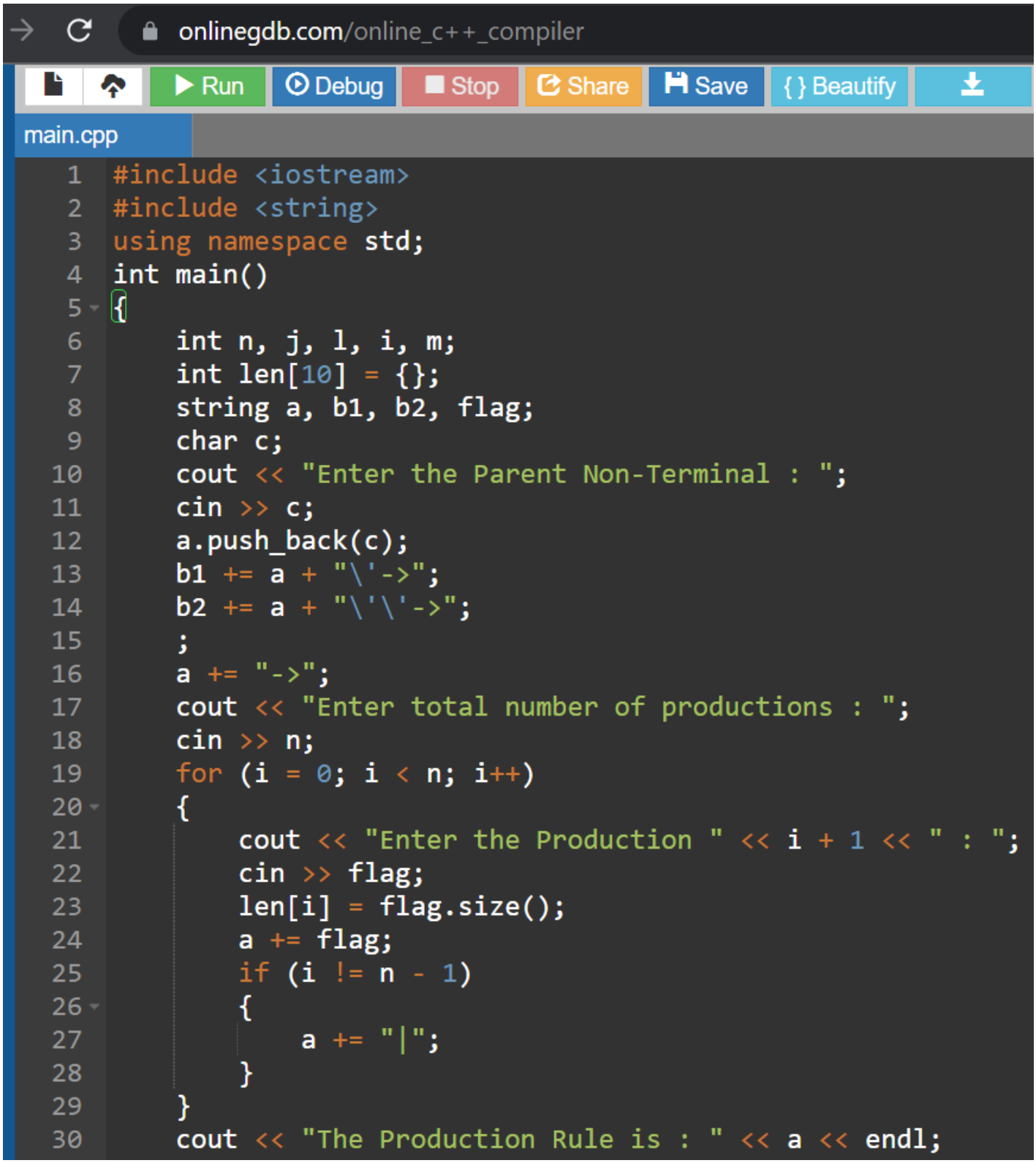
(ii) To remove left factoring

→ For the corner prefixes we must only 1 production

→ So here the corner prefixes can be terminal or a non terminal as it can be a combination of both.

→ The result that is obtained after this process of left recursion & then left factoring is known as left factor grammar.

PROGRAM:




```cpp
#include <iostream>
#include <string>
using namespace std;
int main()
{
    int n, j, l, i, m;
    int len[10] = {};
    string a, b1, b2, flag;
    char c;
    cout << "Enter the Parent Non-Terminal : ";
    cin >> c;
    a.push_back(c);
    b1 += a + "\'->";
    b2 += a + "\'\'->";
    ;
    a += "->";
    cout << "Enter total number of productions : ";
    cin >> n;
    for (i = 0; i < n; i++)
    {
        cout << "Enter the Production " << i + 1 << " : ";
        cin >> flag;
        len[i] = flag.size();
        a += flag;
        if (i != n - 1)
        {
            a += "|";
        }
    }
    cout << "The Production Rule is : " << a << endl;
```

```
    char x = a[3];
    for (i = 0, m = 3; i < n; i++)
    {
        if (x != a[m])
        {
            while (a[m++] != '|')
                ;
        }
        else
        {
            if (a[m + 1] != '|')
            {
                b1 += "|" + a.substr(m + 1, len[i] - 1);
                a.erase(m - 1, len[i] + 1);
            }
            else
            {
                b1 += "#";
                a.insert(m + 1, 1, a[0]);
                a.insert(m + 2, 1, '\'');
                m += 4;
            }
        }
    }
    char y = b1[6];
    for (i = 0, m = 6; i < n - 1; i++)
    {
        if (y == b1[m])
        {
            if (b1[m + 1] != '|')
```

```
            if (b1[m + 1] != '|')
            {
                flag.clear();
                for (int s = m + 1; s < b1.length(); s++)
                {
                    flag.push_back(b1[s]);
                }
                b2 += "|" + flag;
                b1.erase(m - 1, flag.length() + 2);
            }
            else
            {
                b1.insert(m + 1, 1, b1[0]);
                b1.insert(m + 2, 2, '\'');
                b2 += "#";
                m += 5;
            }
        }
    }
    b2.erase(b2.size() - 1);
    cout << "After Left Factoring : " << endl;
    cout << a << endl;
    cout << b1 << endl;
    cout << b2 << endl;
    return 0;
}
```

OUTPUT:

```
Enter the Parent Non-Terminal : M
Enter total number of productions : 4
Enter the Production 1 : i
Enter the Production 2 : iM
Enter the Production 3 : (M)
Enter the Production 4 : iM+M
The Production Rule is : M->i|iM|(M)|iM+M
After Left Factoring :
M->iM'|(M)
M'->#|MM''
M''->#|+M


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

Result : The program for elimination of left recursion and left factoring was executed successsfully in online GDB compiler.

WEEK-5 Computation of FIRST AND FOLLOW

Exp No:-05
Date: 24/02/2022

Pranjay Poddar
RA1911028010129
CSE-CC J1

Computation Of First & Follow

Aim :- To compute First(x) and Follow(x) where x is a grammer symbol.

Procedure :-

(i) For obtaining first :-

(a) If x is a terminal, then First(x) = {x}

(b) If $x \to \varepsilon$ is a production, then add ε to First(x).

(c) If x is a non-terminal and $x \to Y_1 Y_2 \ldots Y_K$ is a production, then add First(Y_1) to First(x). If Y_1 derives ε then add First(Y_2) to First(x).

(ii) For obtaining Follow —

(a) The follow (start symbol) place, \$, where \$ is the input end marker.

(b) If there is a production $A \to \alpha B \beta$, then everything in First(β) except ε is in Follow(B).

(c) If there is a production $A \to \alpha B$ or a production $A \to \alpha B \beta$ where First(β)

contains ε, then everything in follow(A) is in follow(B).

(d) Follow should never contain ε.

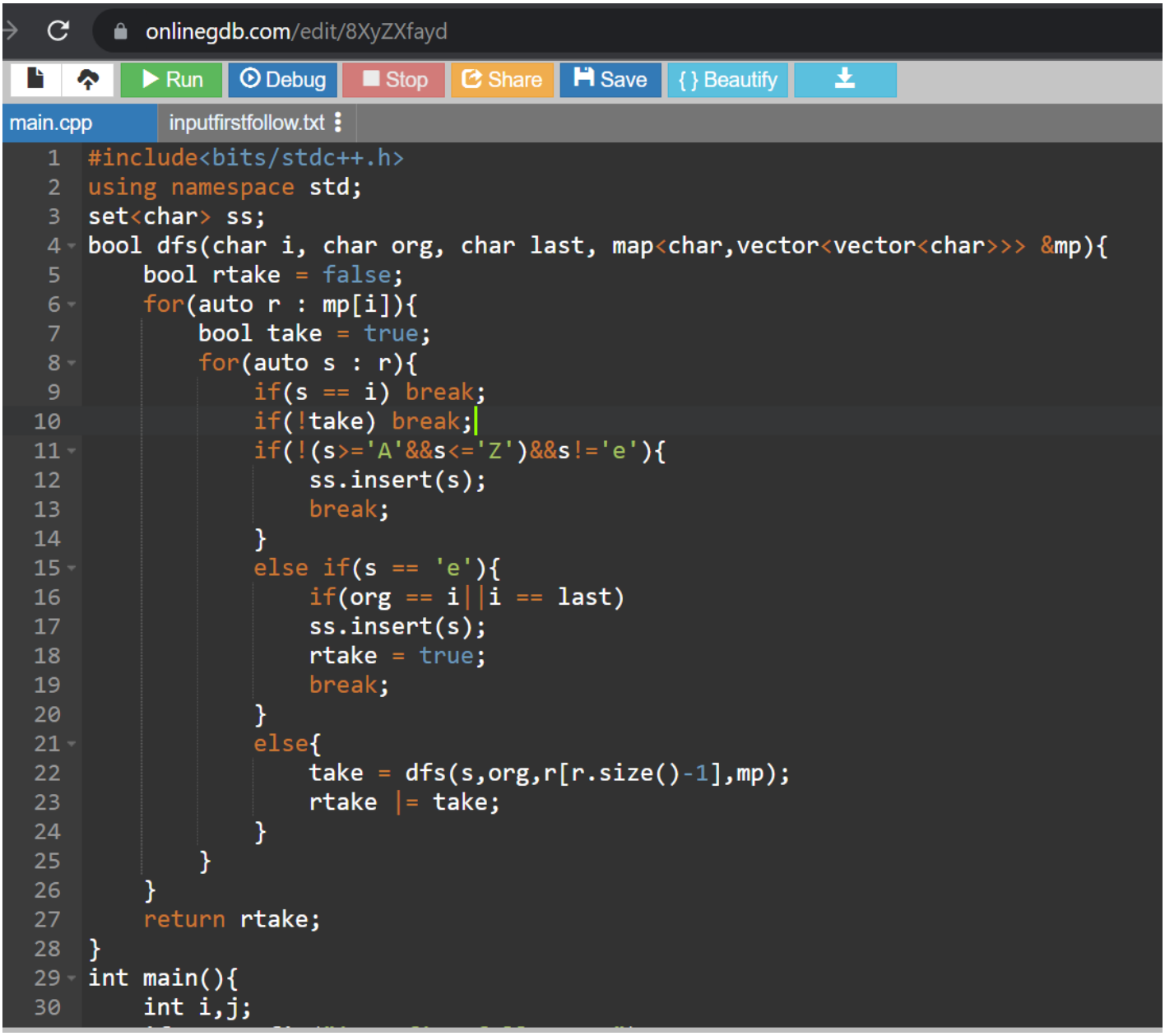
Taking a sample example to prove.

$S \rightarrow Bb \mid cd$

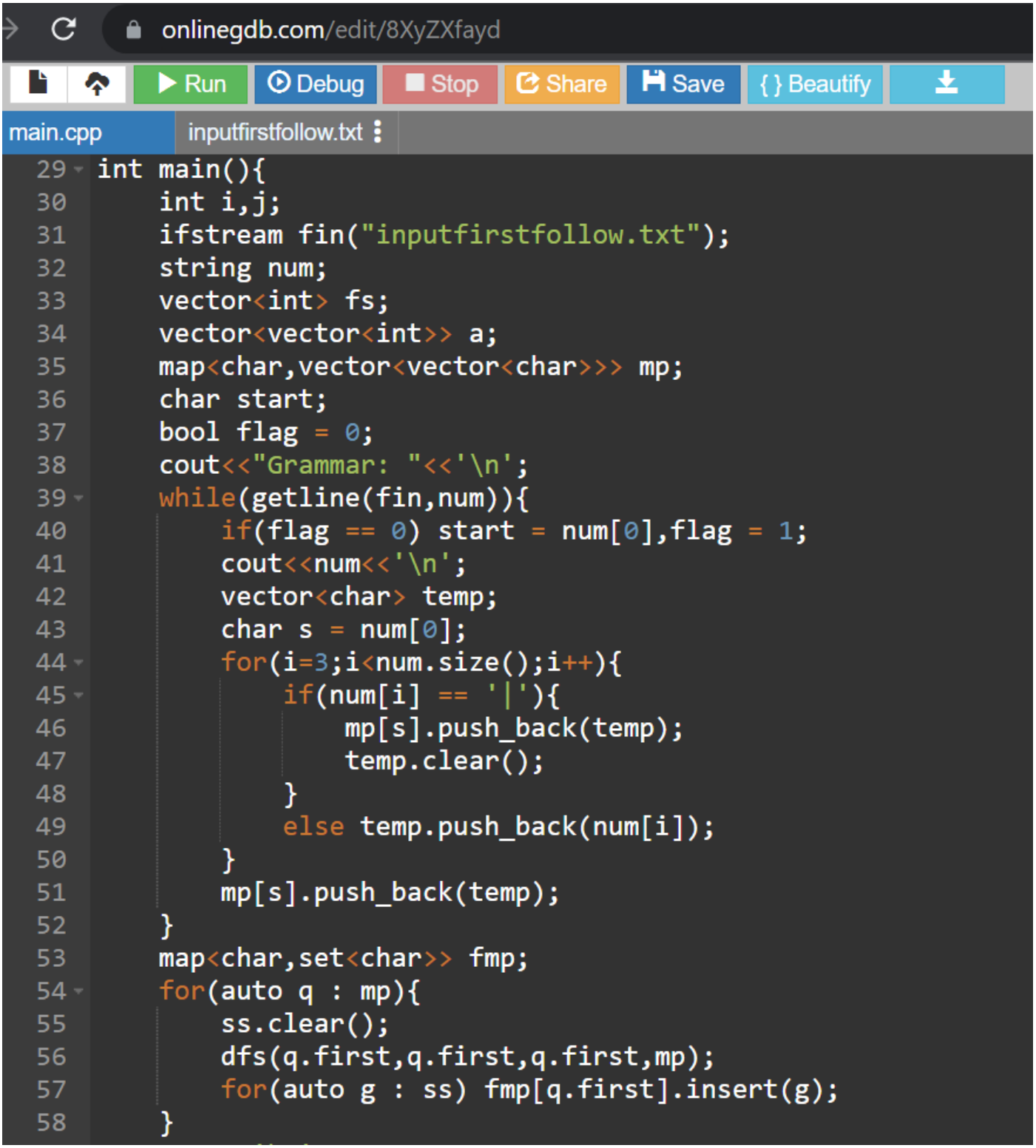
	First	Follow
$S \rightarrow Bb \mid cd$	$\{a, b, c, d\}$	$\{\$\}$
$B \rightarrow aB \mid \varepsilon$	$\{a, \varepsilon\}$	$\{b\}$
$C \rightarrow cC \mid \varepsilon$	$\{c, \varepsilon\}$	$\{d\}$

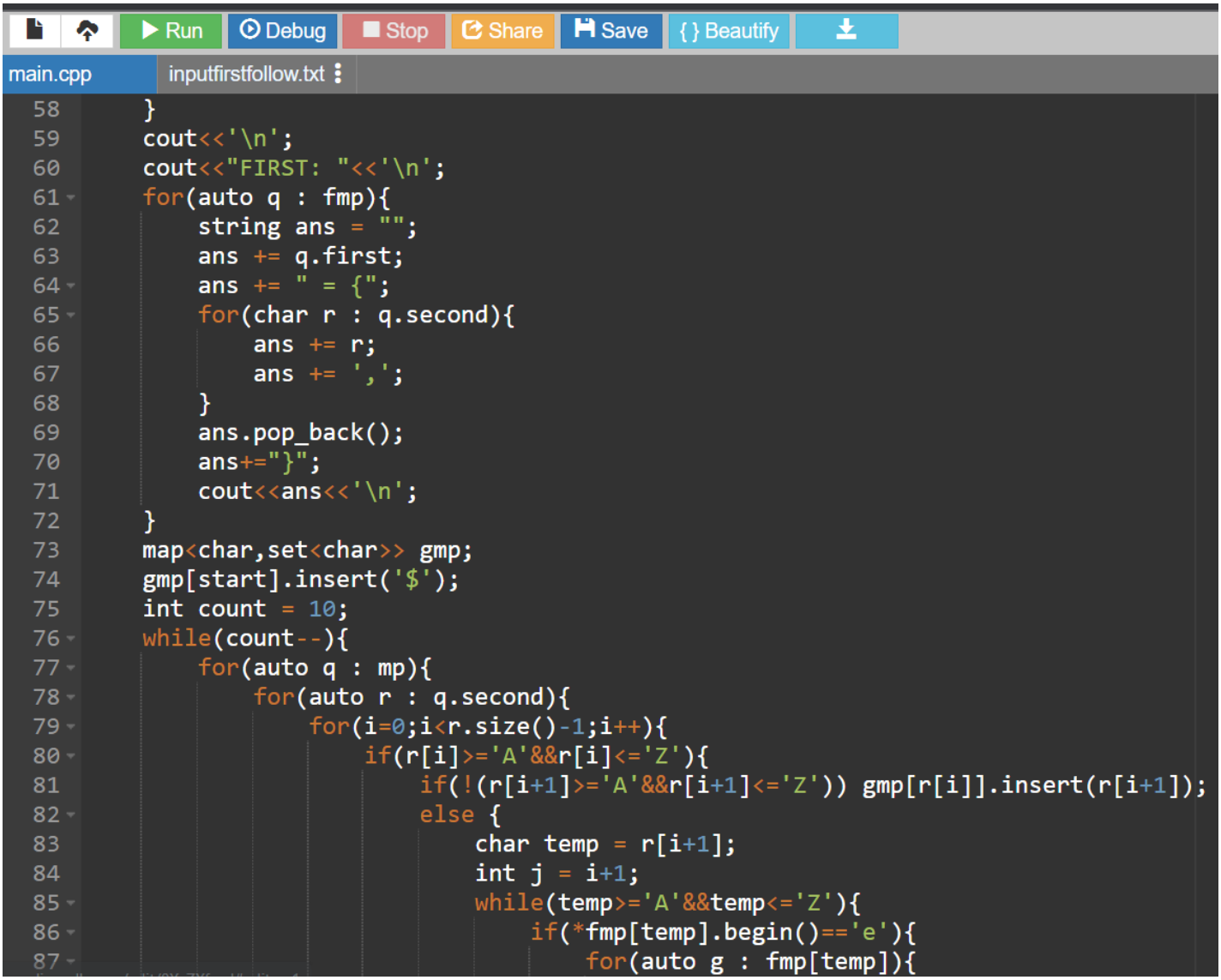
Following the above steps we obtain this.

PROGRAM:




```cpp
#include<bits/stdc++.h>
using namespace std;
set<char> ss;
bool dfs(char i, char org, char last, map<char,vector<vector<char>>> &mp){
    bool rtake = false;
    for(auto r : mp[i]){
        bool take = true;
        for(auto s : r){
            if(s == i) break;
            if(!take) break;
            if(!(s>='A'&&s<='Z')&&s!='e'){
                ss.insert(s);
                break;
            }
            else if(s == 'e'){
                if(org == i||i == last)
                ss.insert(s);
                rtake = true;
                break;
            }
            else{
                take = dfs(s,org,r[r.size()-1],mp);
                rtake |= take;
            }
        }
    }
    return rtake;
}
int main(){
    int i,j;
```


onlinegdb.com/edit/8XyZXfayd
Run
Debug
Stop
Share
Save
{ } Beautify
main.cpp
inputfirstfollow.txt
int main(){
int i,j;
ifstream fin("inputfirstfollow.txt");
string num;
vector<int> fs;
vector<vector<int>> a;
map<char,vector<vector<char>>> mp;
char start;
bool flag = 0;
cout<<"Grammar: "<<'\n';
while(getline(fin,num)){
if(flag == 0) start = num[0],flag = 1;
cout<<num<<'\n';
vector<char> temp;
char s = num[0];
for(i=3;i<num.size();i++){
if(num[i] == '|'){
mp[s].push_back(temp);
temp.clear();
}
else temp.push_back(num[i]);
}
mp[s].push_back(temp);
}
map<char,set<char>> fmp;
for(auto q : mp){
ss.clear();
dfs(q.first,q.first,q.first,mp);
for(auto g : ss) fmp[q.first].insert(g);
}




```
    }
    cout<<'\n';
    cout<<"FIRST: "<<'\n';
    for(auto q : fmp){
        string ans = "";
        ans += q.first;
        ans += " = {";
        for(char r : q.second){
            ans += r;
            ans += ',';
        }
        ans.pop_back();
        ans+="}";
        cout<<ans<<'\n';
    }
    map<char,set<char>> gmp;
    gmp[start].insert('$');
    int count = 10;
    while(count--){
        for(auto q : mp){
            for(auto r : q.second){
                for(i=0;i<r.size()-1;i++){
                    if(r[i]>='A'&&r[i]<='Z'){
                        if(!(r[i+1]>='A'&&r[i+1]<='Z')) gmp[r[i]].insert(r[i+1]);
                        else {
                            char temp = r[i+1];
                            int j = i+1;
                            while(temp>='A'&&temp<='Z'){
                                if(*fmp[temp].begin()=='e'){
                                    for(auto g : fmp[temp]){
```

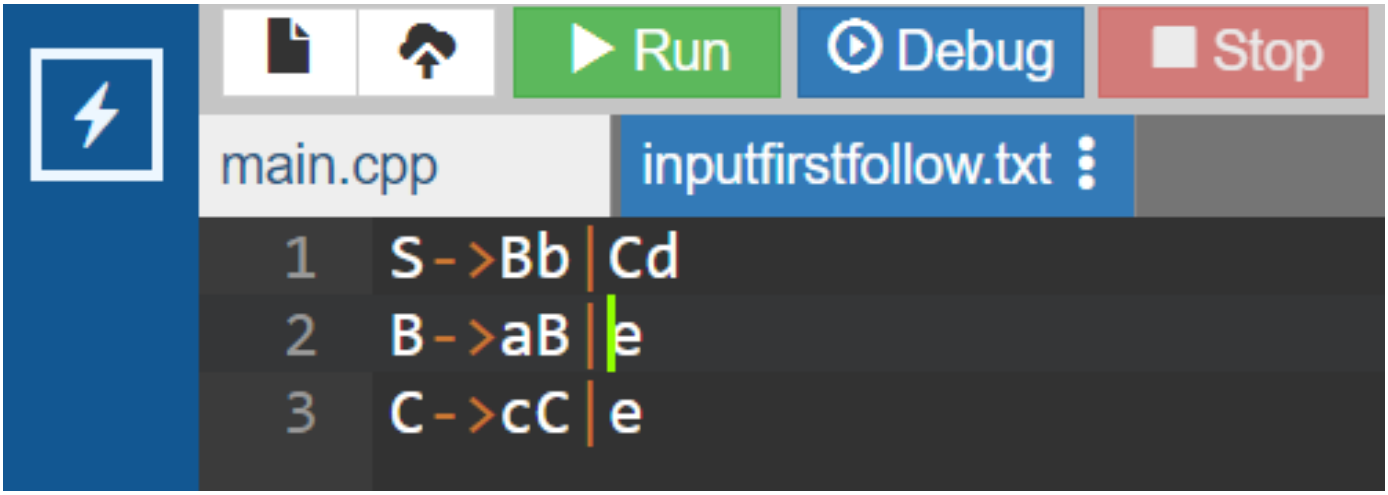
Run Debug Stop Share Save { } Beautify

main.cpp inputfirstfollow.txt

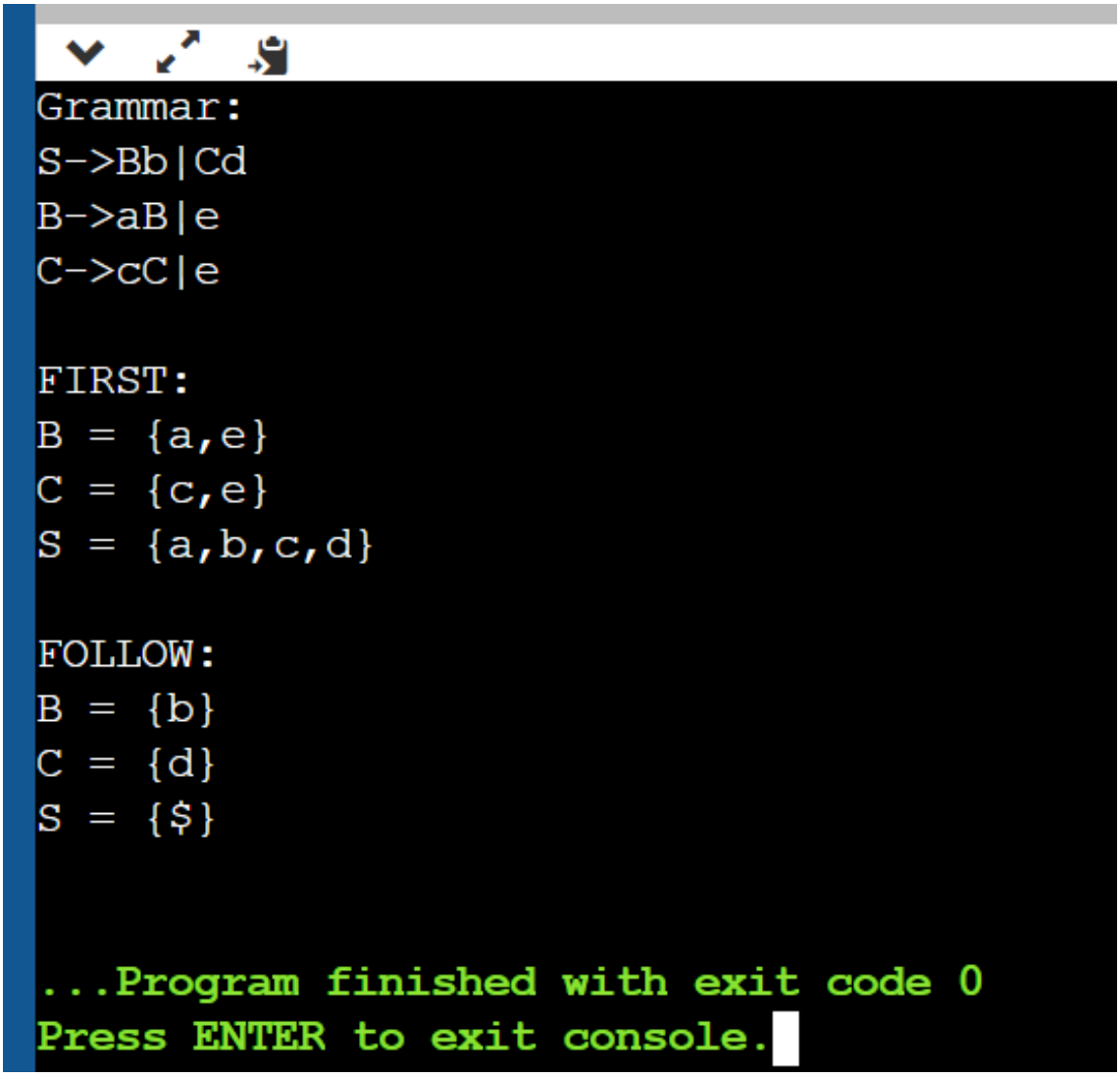
```
                                j++;
                                if(j<r.size()){
                                    temp = r[j];
                                    if(!(temp>='A'&&temp<='Z')){
                                        gmp[r[i]].insert(temp);
                                        break;
                                    }
                                }
                                else{
                                    for(auto g : gmp[q.first]) gmp[r[i]].insert(g);
                                    break;
                                }
                            }
                            else{
                                for(auto g : fmp[temp]){
                                    gmp[r[i]].insert(g);
                                }
                                break;
                            }
                        }
                    }
                }
                if(r[r.size()-1]>='A'&&r[r.size()-1]<='Z'){
                    for(auto g : gmp[q.first]) gmp[r[i]].insert(g);
                }
            }
        }
    }
    cout<<'\n';
    cout<<"FOLLOW: "<<'\n';
```

```
                    for(auto g : gmp[q.first]) gmp[r[i]].insert(g);
                }
            }
        }
    }
    cout<<'\n';
    cout<<"FOLLOW: "<<'\n';
    for(auto q : gmp){
        string ans = "";
        ans += q.first;
        ans += " = {";
        for(char r : q.second){
            ans += r;
            ans += ',';
        }
        ans.pop_back();
        ans+="}";
        cout<<ans<<'\n';
    }
    return 0;
}
```

INPUT:



OUTPUT:



Result : The program for Computation of FIRST AND FOLLOW was executed successsfully in online GDB compiler.

WEEK-6 Construction of Predictive Parsing Table (LL(1))

Experiment-6

Date - 4/9/2022

Pranjay Poddar
RA1911028010129
J1 CSE-CC

Construction Of Predictive Parsing Table

Aim:- To program a code to construct a predictive parsing table for given grammer.

Algorithm:- 1) We need to check if there is left recursive of left following in the grammer.

for eg

$E \rightarrow TE'$

$E' \rightarrow +TE' | \epsilon$

$T \rightarrow FT'$

$T \rightarrow *FT' | \epsilon$

$F \rightarrow id | \epsilon$

2) We need to determine the first() & follow() of all the non-terminals.

Refering to the above example —

	first()	follow()
E	{ id, (}	{ $,) }
E'	{ +, ε }	{ $,) }
T	{ id, (}	{ +, $,) }
T'	{ *, ε }	{ +, $,) }
F	{ id, (}	{ *, +, $ }

(3) For each production if $A \to \alpha$ then,

(1) Find first (α) and for each terminal in first (α), we made an entry $A \to \alpha$ in the table.

(2) If first (α) contains ϵ as di terminal then find follow (A) and make an entry in the table A.

(3) If the first (α) contains ϵ and follow (A) $ a terminal, then make entry $A \to \alpha$ in the for $.

for the example taken →

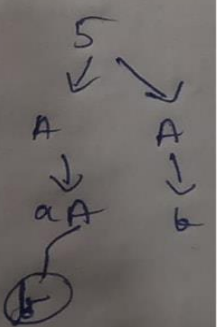
	id	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \epsilon$	$E' \to \epsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \epsilon$	$T' \to *FT'$		$T' \to \epsilon$	$T' \to \epsilon$
F	$F \to id$			$F \to (S)$		

Manual Calculation of Lab Exercise →

$S \to AA$ — (1)

$A \to aA$ — (2)

$A \to b$ — (3)

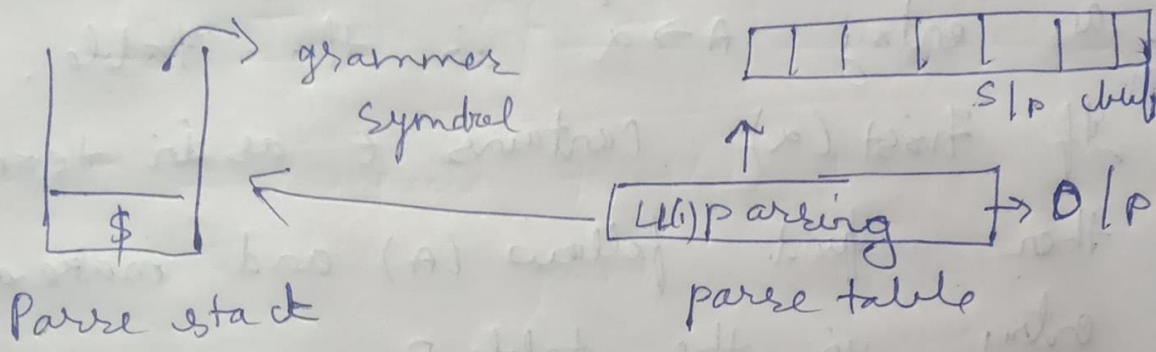


First (S) = {a, b, }

First (A) = {a, b}

follow (S) = {$} [follow of apart is $]

follow (A) = {a, b, $}

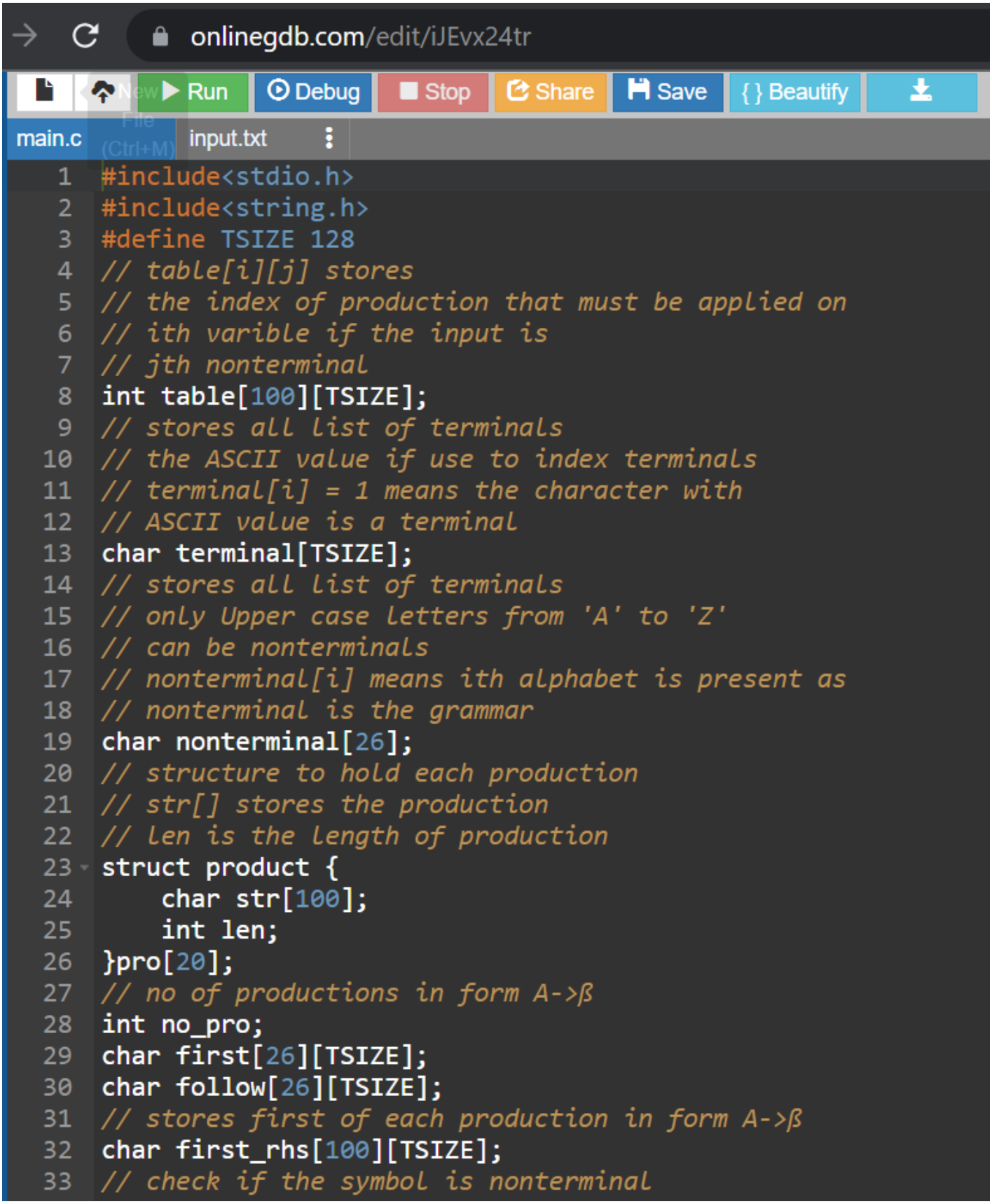


String accepted = abba

String rejected = abab

	a	b	$
S	1	1	
A	2	2	

PROGRAM:


onlinegdb.com/edit/iJEvx24tr
New File (Ctrl+M)
Run
Debug
Stop
Share
Save
{ } Beautify
main.c
input.txt
#include<stdio.h>
#include<string.h>
#define TSIZE 128
// table[i][j] stores
// the index of production that must be applied on
// ith varible if the input is
// jth nonterminal
int table[100][TSIZE];
// stores all list of terminals
// the ASCII value if use to index terminals
// terminal[i] = 1 means the character with
// ASCII value is a terminal
char terminal[TSIZE];
// stores all list of terminals
// only Upper case letters from 'A' to 'Z'
// can be nonterminals
// nonterminal[i] means ith alphabet is present as
// nonterminal is the grammar
char nonterminal[26];
// structure to hold each production
// str[] stores the production
// len is the length of production
struct product {
char str[100];
int len;
}pro[20];
// no of productions in form A->ß
int no_pro;
char first[26][TSIZE];
char follow[26][TSIZE];
// stores first of each production in form A->ß
char first_rhs[100][TSIZE];
// check if the symbol is nonterminal

```
// check if the symbol is nonterminal
int isNT(char c) {
    return c >= 'A' && c <= 'Z';
}
// reading data from the file
void readFromFile() {
    FILE* fptr;
    fptr = fopen("input.txt", "r");
    char buffer[255];
    int i;
    int j;
    while (fgets(buffer, sizeof(buffer), fptr)) {
        printf("%s", buffer);
        j = 0;
        nonterminal[buffer[0] - 'A'] = 1;
        for (i = 0; i < strlen(buffer) - 1; ++i) {
            if (buffer[i] == '|') {
                ++no_pro;
                pro[no_pro - 1].str[j] = '\0';
                pro[no_pro - 1].len = j;
                pro[no_pro].str[0] = pro[no_pro - 1].str[0];
                pro[no_pro].str[1] = pro[no_pro - 1].str[1];
                pro[no_pro].str[2] = pro[no_pro - 1].str[2];
                j = 3;
            }
            else {
                pro[no_pro].str[j] = buffer[i];
                ++j;
                if (!isNT(buffer[i]) && buffer[i] != '-' && buffer[i] != '>') {
                    terminal[buffer[i]] = 1;
                }
            }
        }
        pro[no_pro].len = j;
        ++no_pro;
    }
}
void add_FIRST_A_to_FOLLOW_B(char A, char B) {
    int i;
    for (i = 0; i < TSIZE; ++i) {
        if (i != '^')
            follow[B - 'A'][i] = follow[B - 'A'][i] || first[A - 'A'][i];
    }
}
void add_FOLLOW_A_to_FOLLOW_B(char A, char B) {
    int i;
    for (i = 0; i < TSIZE; ++i) {
        if (i != '^')
            follow[B - 'A'][i] = follow[B - 'A'][i] || follow[A - 'A'][i];
    }
}
void FOLLOW() {
    int t = 0;
    int i, j, k, x;
    while (t++ < no_pro) {
        for (k = 0; k < 26; ++k) {
            if (!nonterminal[k])    continue;
            char nt = k + 'A';
            for (i = 0; i < no_pro; ++i) {
                for (j = 3; j < pro[i].len; ++j) {
                    if (nt == pro[i].str[j]) {
                        for (x = j + 1; x < pro[i].len; ++x) {
                            char sc = pro[i].str[x];
                            if (isNT(sc)) {
                                add_FIRST_A_to_FOLLOW_B(sc, nt);
                                if (first[sc - 'A']['^'])
                                    continue;
                            }
```

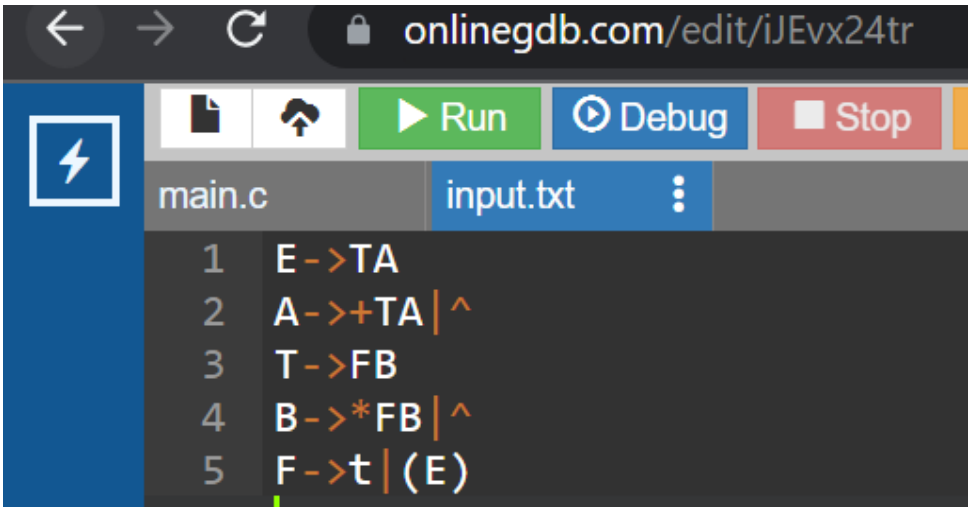
```
                            else {
                                follow[nt - 'A'][sc] = 1;
                            }
                            break;
                        }
                        if (x == pro[i].len)
                            add_FOLLOW_A_to_FOLLOW_B(pro[i].str[0], nt);
                    }
                }
            }
        }
    }
}
void add_FIRST_A_to_FIRST_B(char A, char B) {
    int i;
    for (i = 0; i < TSIZE; ++i) {
        if (i != '^') {
            first[B - 'A'][i] = first[A - 'A'][i] || first[B - 'A'][i];
        }
    }
}
void FIRST() {
    int i, j;
    int t = 0;
    while (t < no_pro) {
        for (i = 0; i < no_pro; ++i) {
            for (j = 3; j < pro[i].len; ++j) {
                char sc = pro[i].str[j];
                if (isNT(sc)) {
                    add_FIRST_A_to_FIRST_B(sc, pro[i].str[0]);
                    if (first[sc - 'A']['^'])
                        continue;
                }
                else {
                    first[pro[i].str[0] - 'A'][sc] = 1;
                }
                break;
            }
            if (j == pro[i].len)
                first[pro[i].str[0] - 'A']['^'] = 1;
        }
        ++t;
    }
}
void add_FIRST_A_to_FIRST_RHS__B(char A, int B) {
    int i;
    for (i = 0; i < TSIZE; ++i) {
        if (i != '^')
            first_rhs[B][i] = first[A - 'A'][i] || first_rhs[B][i];
    }
}
// Calculates FIRST(ß) for each A->ß
void FIRST_RHS() {
    int i, j;
    int t = 0;
    while (t < no_pro) {
        for (i = 0; i < no_pro; ++i) {
            for (j = 3; j < pro[i].len; ++j) {
                char sc = pro[i].str[j];
                if (isNT(sc)) {
                    add_FIRST_A_to_FIRST_RHS__B(sc, i);
                    if (first[sc - 'A']['^'])
                        continue;
                }
                else {
                    first_rhs[i][sc] = 1;
                }
                break;
```

```
169                 }
170                 if (j == pro[i].len)
171                     first_rhs[i]['^'] = 1;
172             }
173             ++t;
174         }
175     }
176     int main() {
177         readFromFile();
178         follow[pro[0].str[0] - 'A']['$'] = 1;
179         FIRST();
180         FOLLOW();
181         FIRST_RHS();
182         int i, j, k;
183 
184         // display first of each variable
185         printf("\n");
186         for (i = 0; i < no_pro; ++i) {
187             if (i == 0 || (pro[i - 1].str[0] != pro[i].str[0])) {
188                 char c = pro[i].str[0];
189                 printf("FIRST OF %c: ", c);
190                 for (j = 0; j < TSIZE; ++j) {
191                     if (first[c - 'A'][j]) {
192                         printf("%c ", j);
193                     }
194                 }
195                 printf("\n");
196             }
197         }
198 
199         // display follow of each variable
200         printf("\n");
201         for (i = 0; i < no_pro; ++i) {
202             if (i == 0 || (pro[i - 1].str[0] != pro[i].str[0])) {
```

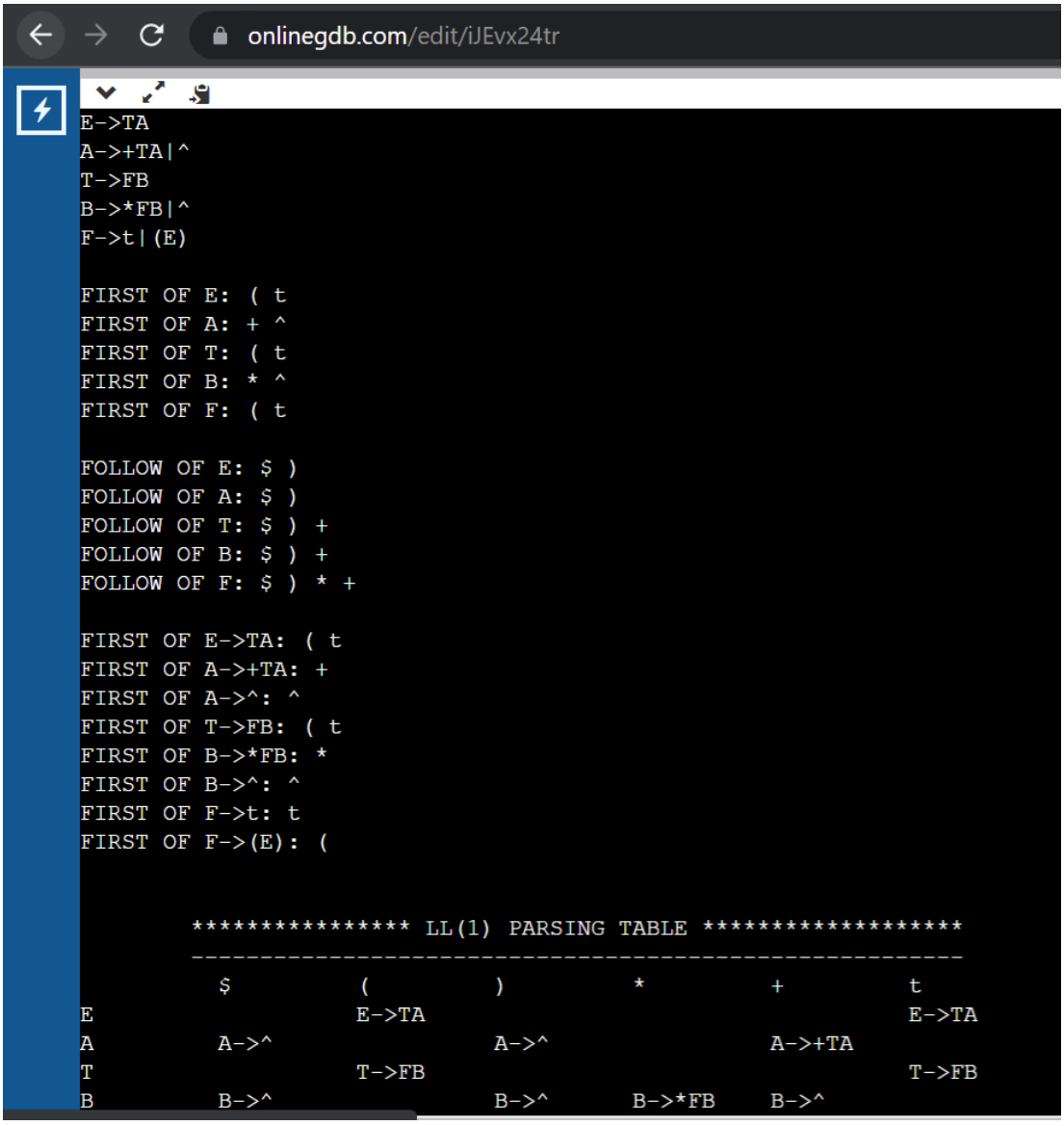
```
203                 char c = pro[i].str[0];
204                 printf("FOLLOW OF %c: ", c);
205                 for (j = 0; j < TSIZE; ++j) {
206                     if (follow[c - 'A'][j]) {
207                         printf("%c ", j);
208                     }
209                 }
210                 printf("\n");
211             }
212         }
213         // display first of each variable ß
214         // in form A->ß
215         printf("\n");
216         for (i = 0; i < no_pro; ++i) {
217             printf("FIRST OF %s: ", pro[i].str);
218             for (j = 0; j < TSIZE; ++j) {
219                 if (first_rhs[i][j]) {
220                     printf("%c ", j);
221                 }
222             }
223             printf("\n");
224         }
225 
226         // the parse table contains '$'
227         // set terminal['$'] = 1
228         // to include '$' in the parse table
229         terminal['$'] = 1;
230 
231         // the parse table do not read '^'
232         // as input
233         // so we set terminal['^'] = 0
234         // to remove '^' from terminals
235         terminal['^'] = 0;
236 
```

```
    // printing parse table
    printf("\n");
    printf("\n\t*************** LL(1) PARSING TABLE *****************\n");
    printf("\t-----------------------------------------------------\n");
    printf("%-10s", "");
    for (i = 0; i < TSIZE; ++i) {
        if (terminal[i])    printf("%-10c", i);
    }
    printf("\n");
    int p = 0;
    for (i = 0; i < no_pro; ++i) {
        if (i != 0 && (pro[i].str[0] != pro[i - 1].str[0]))
            p = p + 1;
        for (j = 0; j < TSIZE; ++j) {
            if (first_rhs[i][j] && j != '^') {
                table[p][j] = i + 1;
            }
            else if (first_rhs[i]['^']) {
                for (k = 0; k < TSIZE; ++k) {
                    if (follow[pro[i].str[0] - 'A'][k]) {
                        table[p][k] = i + 1;
                    }
                }
            }
        }
    }
    k = 0;
    for (i = 0; i < no_pro; ++i) {
        if (i == 0 || (pro[i - 1].str[0] != pro[i].str[0])) {
            printf("%-10c", pro[i].str[0]);
            for (j = 0; j < TSIZE; ++j) {
                if (table[k][j]) {
                    printf("%-10s", pro[table[k][j] - 1].str);
                }
                else if (terminal[j]) {
                    printf("%-10s", "");
                }
            }
            ++k;
            printf("\n");
        }
    }
}
```

INPUT:



OUTPUT:



Result : The program for Construction of Predictive Parsing Table (LL(1)) was executed successsfully in online GDB compiler.

WEEK-7 Implementation of Shift Reduce Parsing

Experiment No.-7
Date : 11-03-2022

Pranjay Poddar
RA1911028010129
CSE-CC J1

Shift Reduce Parser

Aim: To write a program related to shift reduce parser.

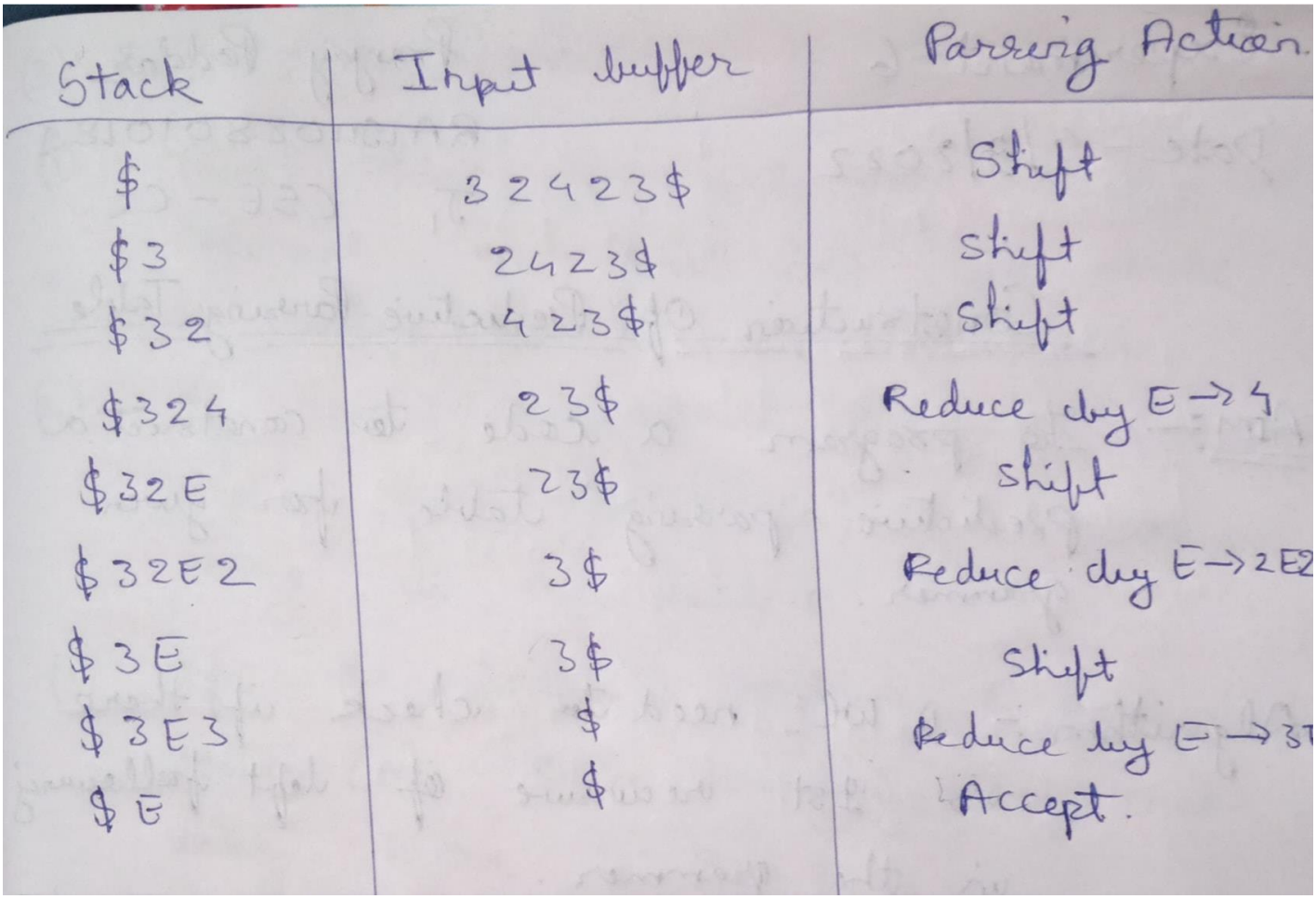
Algorithm :-

(1) Start the program.
(2) Initialize the required variables.
(3) Get the number of productions from the user.
(4) Perform shift operation which involves moving of symbols from input buffer or to the stack.
(5) Perform the reduce operation by popping out the RHS of production rule and pushing the LHS of production.
(6) If the start symbol is present in the stack and the input buffer is empty then accept the parsing action.
(7) When accept action is obtained, it means that successful parsing is done.

Calculation :- Let's consider the grammer.

E → 2E2
E → 3E3
E → 4

INPUT → 32423
String



Stack	Input buffer	Parsing Action
$	32423$	Shift
$3	2423$	shift
$32	423$	Shift
$324	23$	Reduce by E→4
$32E	23$	shift
$32E2	3$	Reduce by E→2E2
$3E	3$	Shift
$3E3	$	Reduce by E→3
$E	$	Accept.

PROGRAM:

onlinegdb.com/edit/fTjmPYt-f

Run | Debug | Stop | Share | Save | { } Beautify

main.cpp

```
#include<bits/stdc++.h>
using namespace std;
struct grammer{
 char p[20];
 char prod[20];
}g[10];
int main()
{
 cout<<"\t\t\t SHIFT REDUCE PARSER\t\t\t\n";
 int i,stpos,j,k,l,m,o,p,f,r;
 int np,tspos,cr;
 cout<<"\nEnter Number of productions:";
 cin>>np;
 char sc,ts[10];
 cout<<"\nEnter productions:\n";
 for(i=0;i<np;i++)
 {
 cin>>ts;
 strncpy(g[i].p,ts,1);
 strcpy(g[i].prod,&ts[3]);
 }
 char ip[10];
 cout<<"\nEnter Input:";
 cin>>ip;
 int lip=strlen(ip);
 char stack[10];

 stpos=0;
 i=0;

 sc=ip[i];
 stack[stpos]=sc;
 i++;stpos++;
```

```
34  cout<<"\n\nStack\t\tInput\t\tAction";
35  do
36  {
37  r=1;
38  while(r!=0)
39  {
40  cout<<"\n";
41  for(p=0;p<stpos;p++)
42  {
43  cout<<stack[p];
44  }
45  cout<<"\t\t";
46  for(p=i;p<lip;p++)
47  {
48  cout<<ip[p];
49  }
50  if(r==2)
51  {
52  cout<<"\t\tReduced";
53  }
54  else
55  {
56  cout<<"\t\tShifted";
57  }
58  r=0;
59
60
61
62  for(k=0;k<stpos;k++)
63  {
64  f=0;
65  for(l=0;l<10;l++)
66  {
67  ts[l]='\0';
```

```
 }
 tspos=0;
 for(l=k;l<stpos;l++)
 {
 ts[tspos]=stack[l];
 tspos++;
 }

 for(m=0;m<np;m++)
 {
 cr = strcmp(ts,g[m].prod);

 if(cr==0)
 {
 for(l=k;l<10;l++)
 {
 stack[l]='\0';
 stpos--;
 }
 stpos=k;

 strcat(stack,g[m].p);
 stpos++;
 r=2;
 }
 }
 }
 }

 sc=ip[i];
 stack[stpos]=sc;
 i++;stpos++;
 }while(strlen(stack)!=1 && stpos!=lip);
 if(strlen(stack)==1)
 {
 cout<<"\n\n \t\t\tSTRING IS ACCEPTED\t\t\t";
 }
 else
cout<<"\n\n \t\t\tSTRING IS REJECTED\t\t\t";
 return 0;
}

```

INPUT:

```
                        SHIFT REDUCE PARSER

Enter Number of productions:4

Enter productions:
E->E+E
E->E*E
E->(E)
E->a

Enter Input:(a*a)+a
```

OUTPUT:

```
Stack           Input           Action
(               a*a)+a          Shifted
(a              *a)+a           Shifted
(E              *a)+a           Reduced
(E*             a)+a            Shifted
(E*a            )+a             Shifted
(E*E            )+a             Reduced
(E              )+a             Reduced
(E)             +a              Shifted
E               +a              Reduced
E+              a               Shifted
E+a                             Shifted
E+E                             Reduced
E                               Reduced

                    STRING IS ACCEPTED

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

Result : The program Implementation of Shift Reduce Parsing was executed successsfully in online GDB compiler.

WEEK-8 Computation of LEADING AND TRAILING

Exp-8
28/03/22

Pranjay Poddar
RA1911028010129
J1 CSE-CC

Computation of Leading & trailing

Aim:- To write a program for computation & construction of leading and trailing

Algorithm:- Algorithm for leading (A).

(1) 'a' is in leading (A) iff. $A \rightarrow \gamma a \delta$ where γ is ϵ or any non-terminal.

(2) If 'a' is in leading (B) & $A \rightarrow B$, then 'a' is in leading (A).

Algorithm for trailing (A)

(1) 'a' is in trailing (A) iff $(A] \rightarrow \gamma a \delta$ where δ is ϵ or any non terminal

(2) If 'a' is in terminal trailing (B] & A $A \rightarrow B$, then 'a' is in trailing (A].

No. of Productions = 2

Variable = S, T

No of terminals = 5

Terminals are = (,), *, +, ,

S's production -

S → S+T | (T)

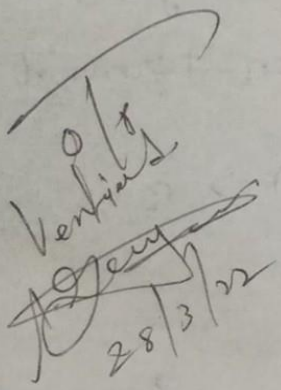
T → *, ,

Lead (S) = { (, +, *, , }

Lead (T) = { *, , }

Trail (S) = {), +, *, , }

Trail (T) = { *, , }



PROGRAM:

main.cpp

```
#include<iostream>
#include<conio.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
using namespace std;

int vars,terms,i,j,k,m,rep,count,temp=-1;
char var[10],term[10],lead[10][10],trail[10][10];
struct grammar
{
    int prodno;
    char lhs,rhs[20][20];
}gram[50];
void get()
{
    cout<<"\nLEADING AND TRAILING\n";
    cout<<"\nEnter the no. of variables : ";
    cin>>vars;
    cout<<"\nEnter the variables : \n";
    for(i=0;i<vars;i++)
    {
        cin>>gram[i].lhs;
        var[i]=gram[i].lhs;
    }
    cout<<"\nEnter the no. of terminals : ";
    cin>>terms;
    cout<<"\nEnter the terminals : ";
    for(j=0;j<terms;j++)
        cin>>term[j];
    cout<<"\nPRODUCTION DETAILS\n";
    for(i=0;i<vars;i++)
    {
        cout<<"\nEnter the no. of production of "<<gram[i].lhs<<":";
```

```
35          cin>>gram[i].prodno;
36          for(j=0;j<gram[i].prodno;j++)
37          {
38              cout<<gram[i].lhs<<"->";
39              cin>>gram[i].rhs[j];
40          }
41      }
42  }
43  void leading()
44  {
45      for(i=0;i<vars;i++)
46      {
47          for(j=0;j<gram[i].prodno;j++)
48          {
49              for(k=0;k<terms;k++)
50              {
51                  if(gram[i].rhs[j][0]==term[k])
52                      lead[i][k]=1;
53                  else
54                  {
55                      if(gram[i].rhs[j][1]==term[k])
56                          lead[i][k]=1;
57                  }
58              }
59          }
60      }
61      for(rep=0;rep<vars;rep++)
62      {
63          for(i=0;i<vars;i++)
64          {
65              for(j=0;j<gram[i].prodno;j++)
66              {
67                  for(m=1;m<vars;m++)
68                  {
```

```
                    if(gram[i].rhs[j][0]==var[m])
                    {
                        temp=m;
                        goto out;
                    }
                }
                out:
                for(k=0;k<terms;k++)
                {
                    if(lead[temp][k]==1)
                        lead[i][k]=1;
                }
            }
        }
    }
}
void trailing()
{
    for(i=0;i<vars;i++)
    {
        for(j=0;j<gram[i].prodno;j++)
        {
            count=0;
            while(gram[i].rhs[j][count]!='\x0')
                count++;
            for(k=0;k<terms;k++)
            {
                if(gram[i].rhs[j][count-1]==term[k])
                    trail[i][k]=1;
                else
                {
                    if(gram[i].rhs[j][count-2]==term[k])
                        trail[i][k]=1;
                }
```

```
            }
        }
    }
    for(rep=0;rep<vars;rep++)
    {
        for(i=0;i<vars;i++)
        {
            for(j=0;j<gram[i].prodno;j++)
            {
                count=0;
                while(gram[i].rhs[j][count]!='\x0')
                    count++;
                for(m=1;m<vars;m++)
                {
                    if(gram[i].rhs[j][count-1]==var[m])
                        temp=m;
                }
                for(k=0;k<terms;k++)
                {
                    if(trail[temp][k]==1)
                        trail[i][k]=1;
                }
            }
        }
    }
}
void display()
{
    for(i=0;i<vars;i++)
    {
        cout<<"\nLEADING("<<gram[i].lhs<<") = ";
        for(j=0;j<terms;j++)
        {
            if(lead[i][j]==1)
```

```
                cout<<term[j]<<",";
            }
        }
        cout<<endl;
        for(i=0;i<vars;i++)
        {
            cout<<"\nTRAILING("<<gram[i].lhs<<") = ";
            for(j=0;j<terms;j++)
            {
                if(trail[i][j]==1)
                    cout<<term[j]<<",";
            }
        }
}
int main()
{

    get();
    leading();
    trailing();
    display();

}
```

OUTPUT:

```
LEADING AND TRAILING

Enter the no. of variables : 3

Enter the variables :
E
T
F

Enter the no. of terminals : 5

Enter the terminals : )
(
*
+
i

PRODUCTION DETAILS

Enter the no. of production of E:2
E->E+T
E->T

Enter the no. of production of T:2
T->T*F
T->F

Enter the no. of production of F:2
F->(E)
F->i

LEADING(E) = (,*,+,i,
LEADING(T) = (,*,i,
LEADING(F) = (,i,
```

```
TRAILING(E) = ),*,+,i,
TRAILING(T) = ),*,i,
TRAILING(F) = ),i,
```

Result : The program Implementation of Computation of leading and trailing was executed successsfully in online GDB compiler.

WEEK-8 Computation of LR(0) items

Exp - 9
28/03/22

Pranjay Poddar
RA1911028010129
CSE-CC J1

Computing of LR(0)

Aim:- To produce argumented grammer and get closure to obtain LR(0).

Algorithm:-

1. Create argumented grammer.
2. After that start reading closure by right shift dot operator.
3. Keep on creating closures till all the states or productions are reduced.
4. Finally after that create an action and goto table where action specific the non-terminals and goto specific terminals.
5. Create the stack implementation for the table and put all the states & reduction in the stack.
6. Take the input string check for it.

Example.

$S \rightarrow AA$
$A \rightarrow aA \mid b$

$S' \to \cdot S$ $\quad A \to \cdot aA$

$S \to \cdot AA$ $\quad A \to \cdot b$

$I_0 =$ closure $(S' \to \cdot S)$

$= S' \to \cdot S$

$S \to \cdot AA$

$A \to \cdot aA$

$A \to \cdot b$

$I_1 = (I_0, S) =$ closure$(S' \to S \cdot) = S' \to S \cdot$

$I_2 = (S_0, A) =$ closure $(S \to A \cdot A)$

$I_3 = (I_2, a) =$ closure $(A \to a \cdot A)$

$A \to a \cdot A \quad A \to \cdot aA \quad A \to \cdot b$

$I_4 = (I_0, b) =$ closure $(A \to b) = A \to b$

$I_5 = (I_2, A) =$ closure $(S \to A \cdot A) = S \to A_2$

$I_6 = (I_3, A) =$ closure $(A \to aA) = A \to aA$

It contains 7 states.

LR(0) Table

States	Actions			Gotos	
	a	b	$	A	S
I_0	S_3	S_4		2	1
I_1			accept		1
I_2	S_3	S_4		5	
I_3	S_3	S_4		6	
I_4	r_3	r_3	r_3		
I_5	r_1	r_1	r_1		
I_6	r_2	r_2	r_2		

PROGRAM:

main.cpp

```
#include<iostream>
#include<conio.h>
#include<string.h>

using namespace std;

char prod[20][20],listofvar[26]="ABCDEFGHIJKLMNOPQR";
int novar=1,i=0,j=0,k=0,n=0,m=0,arr[30];
int noitem=0;

struct Grammar
{
    char lhs;
    char rhs[8];
}g[20],item[20],clos[20][10];

int isvariable(char variable)
{
    for(int i=0;i<novar;i++)
        if(g[i].lhs==variable)
            return i+1;
    return 0;
}
void findclosure(int z, char a)
{
    int n=0,i=0,j=0,k=0,l=0;
    for(i=0;i<arr[z];i++)
    {
        for(j=0;j<strlen(clos[z][i].rhs);j++)
        {
            if(clos[z][i].rhs[j]=='.' && clos[z][i].rhs[j+1]==a)
```

```
            {
                clos[noitem][n].lhs=clos[z][i].lhs;
                strcpy(clos[noitem][n].rhs,clos[z][i].rhs);
                char temp=clos[noitem][n].rhs[j];
                clos[noitem][n].rhs[j]=clos[noitem][n].rhs[j+1];
                clos[noitem][n].rhs[j+1]=temp;
                n=n+1;
            }
        }
    }
    for(i=0;i<n;i++)
    {
        for(j=0;j<strlen(clos[noitem][i].rhs);j++)
        {
            if(clos[noitem][i].rhs[j]=='.' && isvariable(clos[noitem][i].rhs[j+1])>0)
            {
                for(k=0;k<novar;k++)
                {
                    if(clos[noitem][i].rhs[j+1]==clos[0][k].lhs)
                    {
                        for(l=0;l<n;l++)
                            if(clos[noitem][l].lhs==clos[0][k].lhs && strcmp(clos[noitem][l].rhs,clos[0][k].rhs)==0)
                                break;
                        if(l==n)
                        {
                            clos[noitem][n].lhs=clos[0][k].lhs;
                        strcpy(clos[noitem][n].rhs,clos[0][k].rhs);
                            n=n+1;
                        }
                    }
                }
            }
        }
    }
```

```
    int flag=0;
    for(i=0;i<noitem;i++)
    {
        if(arr[i]==n)
        {
            for(j=0;j<arr[i];j++)
            {
                int c=0;
                for(k=0;k<arr[i];k++)
                    if(clos[noitem][k].lhs==clos[i][k].lhs && strcmp(clos[noitem][k].rhs,clos[i][k].rhs)==0)
                        c=c+1;
                if(c==arr[i])
                {
                    flag=1;
                    goto exit;
                }
            }
        }
    }
    exit:;
    if(flag==0)
        arr[noitem++]=n;
}

int main()
{
    cout<<"ENTER THE PRODUCTIONS OF THE GRAMMAR(0 TO END) :\n";
    do
    {
        cin>>prod[i++];
    }while(strcmp(prod[i-1],"0")!=0);
    for(n=0;n<i-1;n++)
    {
        m=0;
```

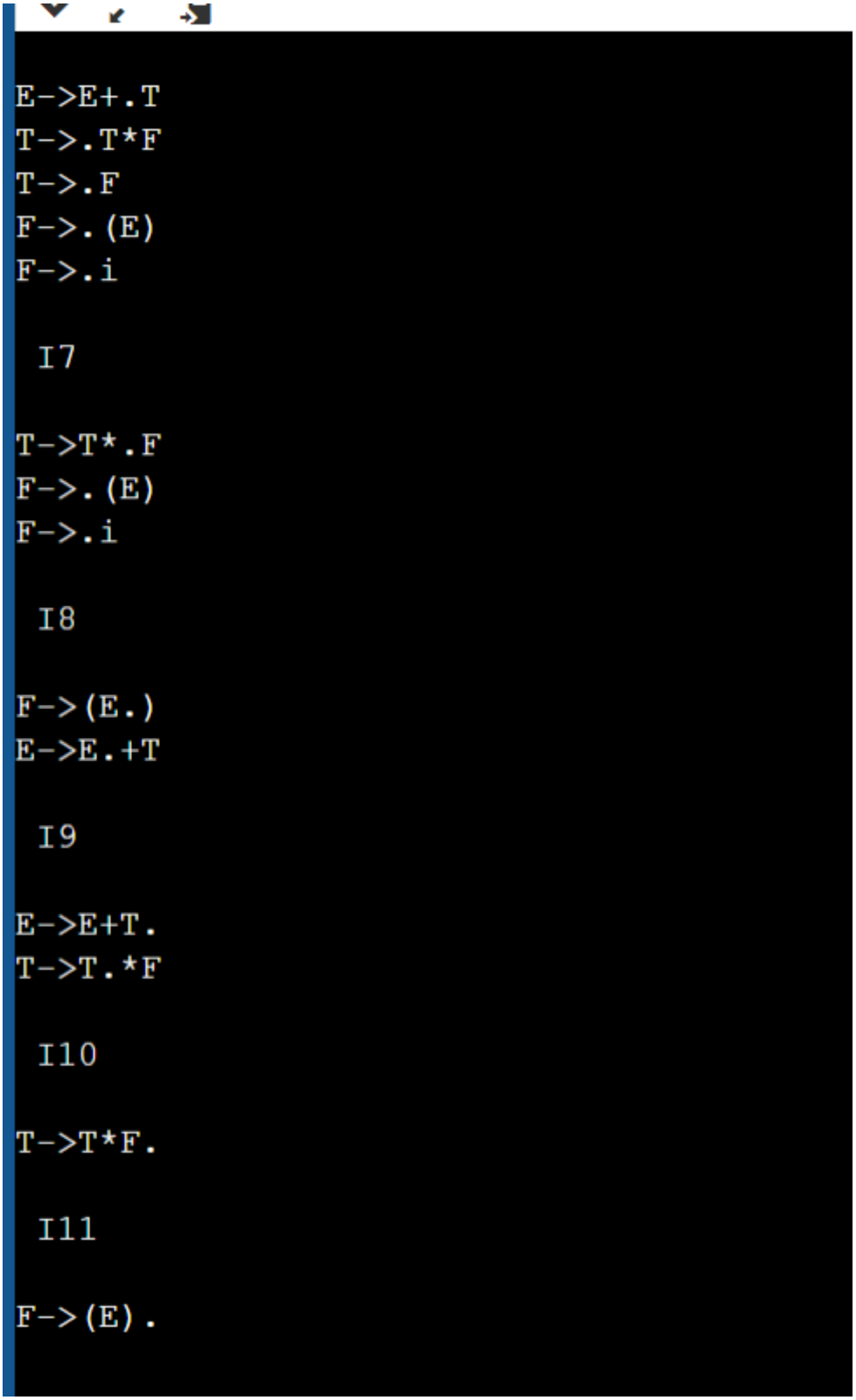
```
        j=novar;
        g[novar++].lhs=prod[n][0];
        for(k=3;k<strlen(prod[n]);k++)
        {
            if(prod[n][k] != '|')
            g[j].rhs[m++]=prod[n][k];
            if(prod[n][k]=='|')
            {
                g[j].rhs[m]='\0';
                m=0;
                j=novar;
                g[novar++].lhs=prod[n][0];
            }
        }
    }
    for(i=0;i<26;i++)
        if(!isvariable(listofvar[i]))
            break;
    g[0].lhs=listofvar[i];
    char temp[2]={g[1].lhs,'\0'};
    strcat(g[0].rhs,temp);
    cout<<"\n\n augumented grammar \n";
    for(i=0;i<novar;i++)
        cout<<endl<<g[i].lhs<<"->"<<g[i].rhs<<" ";

    for(i=0;i<novar;i++)
    {
        clos[noitem][i].lhs=g[i].lhs;
        strcpy(clos[noitem][i].rhs,g[i].rhs);
        if(strcmp(clos[noitem][i].rhs,"ε")==0)
            strcpy(clos[noitem][i].rhs,".");
        else
        {
            for(int j=strlen(clos[noitem][i].rhs)+1;j>=0;j--)
```

```
                clos[noitem][i].rhs[j]=clos[noitem][i].rhs[j-1];
            clos[noitem][i].rhs[0]='.';
        }
    }
    arr[noitem++]=novar;
    for(int z=0;z<noitem;z++)
    {
        char list[10];
        int l=0;
        for(j=0;j<arr[z];j++)
        {
            for(k=0;k<strlen(clos[z][j].rhs)-1;k++)
            {
                if(clos[z][j].rhs[k]=='.')
                {
                    for(m=0;m<l;m++)
                        if(list[m]==clos[z][j].rhs[k+1])
                            break;
                    if(m==l)
                        list[l++]=clos[z][j].rhs[k+1];
                }
            }
        }
        for(int x=0;x<l;x++)
            findclosure(z,list[x]);
    }
    cout<<"\n THE SET OF ITEMS ARE \n\n";
    for(int z=0; z<noitem; z++)
    {
        cout<<"\n I"<<z<<"\n\n";
        for(j=0;j<arr[z];j++)
            cout<<clos[z][j].lhs<<"->"<<clos[z][j].rhs<<"\n";

    }
```

OUTPUT:

```
ENTER THE PRODUCTIONS OF THE GRAMMAR(0 TO END) :
E->E+T
E->T
T->T*F
T->F
F->(E)
F->i
0


 augumented grammar

A->E
E->E+T
E->T
T->T*F
T->F
F->(E)
F->i
 THE SET OF ITEMS ARE


 I0

A->.E
E->.E+T
E->.T
T->.T*F
T->.F
F->.(E)
F->.i
```

```
 I1

A->E.
E->E.+T

 I2

E->T.
T->T.*F

 I3

T->F.

 I4

F->(.E)
E->.E+T
E->.T
T->.T*F
T->.F
F->.(E)
F->.i

 I5

F->i.

 I6

E->E+.T
T->.T*F
```



Result : The program Implementation of Computation of LR(0) items was executed successsfully in online GDB compiler.

WEEK-10 Intermediate code generation – Postfix, Prefix

4/04/22
Ex-10

Pranjay Poddar
RA1911028010129
CSE-CC J1

Intermediate Code Generation: Postfix & Prefix

Aim:- To write a program to implement code generation - Postfix, Prefix.

Procedure:- (1) Declare set of operators.
(2) Initiate an empty stack.
(3) To convert INFIX to POSTFIX follow the following steps.
(4) Scan the infix expression.
(5) If the scanned character is operand. output.
(6) Else, if the procedence of the scanned operator is greater than the precedence of the operator in the stack.
(7) Else, pop all the operators from the stack which are greater than or equal to in precedence.
(8) If the scanned character is an ')' pop to stack.
(9) If the scanned character is an '(' push to the stack.

(10) Pop & output it until a '(' is encountered and discard both the parentheis.

(11) Convert logic of infix to postfix follow the following steps.

(12) First, reverse the infix expression given in the problem.

(13) Scan the expression.

(14) If the operator moves and the stack is formed to be empty then empty push the the operator into the stack.

(15) Repeat steps 6 to 9 until the stack is empty.

Manual Calculation :-

Infix to Prefix

Infix	Prefix
C+D*E	CDE*+

Infix to Postfix

Infix	Postfix
C+D*E	+C*DE

Result: Thus we have implemented a program to generate a postfix and prefix.

PROGRAM:

Postfix:

Prefix:

main.cpp

```
1  //Prefix:
2
3  #include <bits/stdc++.h>
4  using namespace std;
5
6  bool isOperator(char c)
7  {
8  return (!isalpha(c) && !isdigit(c));
9  }
10
11 int getPriority(char C)
12 {
13 if (C == '-' || C == '+')
14 return 1;
15 else if (C == '*' || C == '/')
16 return 2;
17 else if (C == '^')
18 return 3;
19 return 0;
20 }
21
22 string infixToPostfix(string infix)
23 {
24 infix = '(' + infix + ')';
25 int l = infix.size();
26 stack<char> char_stack;
27 string output;
28
29 for (int i = 0; i < l; i++) {
30
31
32 if (isalpha(infix[i]) || isdigit(infix[i]))
33 output += infix[i];
```

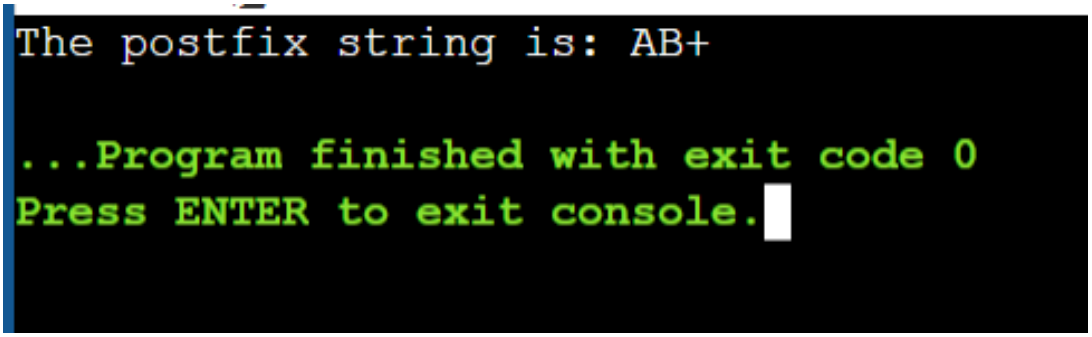
```
35  else if (infix[i] == '(')
36  char_stack.push('(');
37
38
39  else if (infix[i] == ')') {
40  while (char_stack.top() != '(') {
41  output += char_stack.top();
42  char_stack.pop();
43  }
44
45
46  char_stack.pop();
47  }
48
49
50  else
51  {
52  if (isOperator(char_stack.top()))
53  {
54  if(infix[i] == '^')
55  {
56  while (getPriority(infix[i]) <= getPriority(char_stack.top()))
57  {
58  output += char_stack.top();
59  char_stack.pop();
60  }
61
62  }
63  else
64  {
65  while (getPriority(infix[i]) < getPriority(char_stack.top()))
66  {
67  output += char_stack.top();
68  char_stack.pop();
```

```
69  }
70
71  }
72
73
74  char_stack.push(infix[i]);
75  }
76  }
77  }
78  while(!char_stack.empty()){
79  output += char_stack.top();
80  char_stack.pop();
81  }
82  return output;
83  }
84
85  string infixToPrefix(string infix)
86  {
87  int l = infix.size();
88
89
90  reverse(infix.begin(), infix.end());
91
92
93  for (int i = 0; i < l; i++) {
94
95  if (infix[i] == '(') {
96  infix[i] = ')';
97  i++;
98  }
99  else if (infix[i] == ')') {
100 infix[i] = '(';
101 i++;
102 }
```

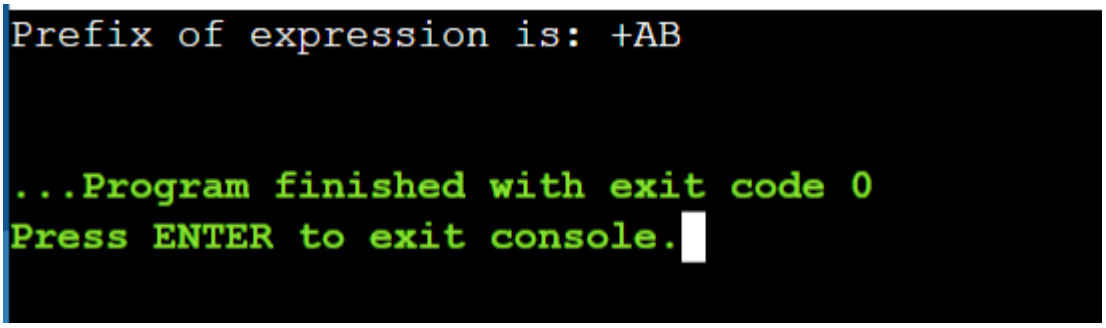
```
113 int main()
114 {
115 string s = ("A+B");
116 cout << "Prefix of expression is: " << infixToPrefix(s) << std::endl;
117 return 0;
118 }
```

OUTPUT:

Postfix:



Prefix:



Result : The program Implementation of Intermediate code generation – Postfix, Prefix was executed successsfully in online GDB compiler.

WEEK-11 Intermediate code generation – Quadruple, Triple, Indirect triple

Experiment - 11
Date:- 4/4/11

Pranjay Poddar
RA1911028010129
CSE - CC J1

Intermediate Code Generator :- Quadruple, Triple Indirect Triple

Aim :- To implement the code related to quadruple, triple & Indirect triple.

Algorithm :- (1) Start

(2) It is a sequence of three-address statements as input.

(3) For each three address statements of the form a :: b c perform the various actions. These are as follows :-

(i) Invoke :- a function getreg to find the location L where the result of computation b op c should be stored.

(2) Consult the address description for y to determine the value of y.

(3) Generate the instructions op z', L where z is used to show the current location of z. If z is both then prefer a register to a memory location.

(iv) If the current value of y & z have no next uses or live on exit from the block.

Calculations :-

Let the expression be : a + b * c + d is

* * $T_1 = B * C$

$T_2 = A + T_2$

$T_3 = T_3 + D * *$

=> a = b + c - d

$t_1 = b + c$

$t_2 = t_1 - d$

a = t 2

Result = Successfully executed the program that depicted the three address code (quadruple, triple, indirect triple).

PROGRAM:

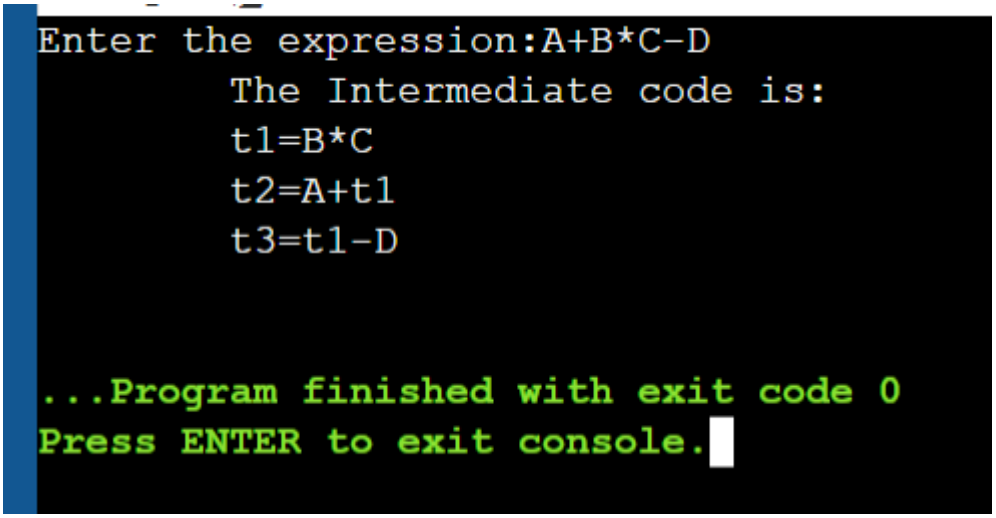
main.c

```c
#include<stdio.h>
#include<ctype.h>
#include<stdlib.h>
#include<string.h>
void small();
void dove(int i);
int p[5]={0,1,2,3,4},c=1,i,k,l,m,pi;
char sw[5]={'=','-','+','/','*'},j[20],a[5],b[5],ch[2];
void main()
{
printf("Enter the expression:");
scanf("%s",j);
printf("\tThe Intermediate code is:\n");
small();
}
void dove(int i)
{
a[0]=b[0]='\0';
if(!isdigit(j[i+2])&&!isdigit(j[i-2]))
{
a[0]=j[i-1];
b[0]=j[i+1];
}
if(isdigit(j[i+2])){
a[0]=j[i-1];
b[0]='t';
b[1]=j[i+2];
}
if(isdigit(j[i-2]))
{
b[0]=j[i+1];
a[0]='t';
a[1]=j[i-2];
b[1]='\0';
```

```
}
if(isdigit(j[i+2]) &&isdigit(j[i-2]))
{
a[0]='t';
b[0]='t';
a[1]=j[i-2];
b[1]=j[i+2];
sprintf(ch,"%d",c);
j[i+2]=j[i-2]=ch[0];
}
if(j[i]=='*')
printf("\tt%d=%s*%s\n",c,a,b);
if(j[i]=='/')
printf("\tt%d=%s/%s\n",c,a,b);
if(j[i]=='+')
printf("\tt%d=%s+%s\n",c,a,b);if(j[i]=='-')
printf("\tt%d=%s-%s\n",c,a,b);
if(j[i]=='=')
printf("\t%c=t%d",j[i-1],--c);
sprintf(ch,"%d",c);
j[i]=ch[0];
c++;
small();
}
void small()
{
pi=0;l=0;
for(i=0;i<strlen(j);i++)
{
for(m=0;m<5;m++)
if(j[i]==sw[m])
if(pi<=p[m])
{
pi=p[m];
```

```
l=1;
k=i;
}
}
if(l==1)
dove(k);
else
exit(0);}
```

OUTPUT:



Result : The program Implementation of Intermediate code generation – Quadruple, Triple, Indirect triple was executed successsfully in online GDB compiler.

WEEK-12 Implementation Of DAG

Exp 12

Date :- 11/04/22

Pranjay Poddar
RA19 11028010129
J1 Section

Implementation of DAG

Aim:- A program to implement DAG.

Procedure :-

(1) The leaves of a graph are labled by unique identity.

(2) Interior nodes of the graph are loaded by an operator symbol.

(3) Nodes are also given sequence of identifiers for lables to store the computed value.

(4) If y operand is undefined then create node (y).

(5) If z operand is underfined then for case (1) create node (z).

(6) For case (i), create node (OP) whose right child is node (z) and left child [] is node (y).

(7) For case (ii) check wether there is node (OP) with one child node (y).

(8) For case (iii) node n will be node (y).

(9) For node (x) delete x from the list of identifiers.

(10) Append a to attach identifiers for the node n found in step-2 finally set node (x) to n.

Calculations :-

Three possible scenarios for building a DAG

Case 1 : x = y or z
Case 2 : x = op y
Case 3 : x = y

Let the expression is a = b + -c + b * -c

z = b *
y = c + z
x = c - y
w = x - c
a = w a = $

O/P verified. 11/4/24

Result : Successfully executed the program related to DAG.

PROGRAM:

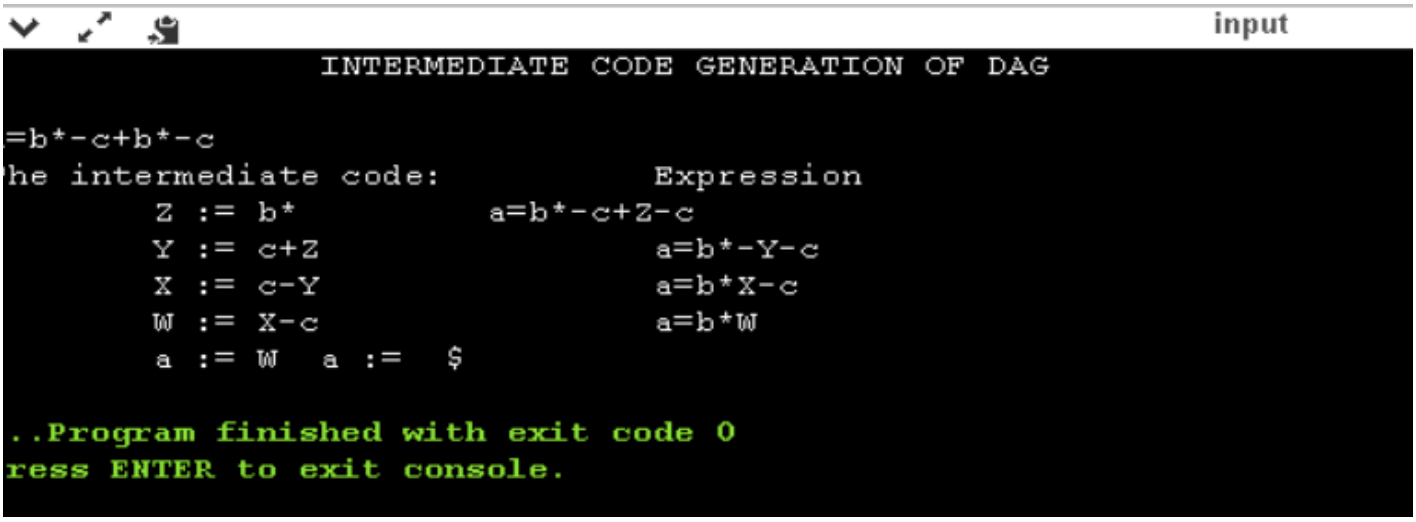
main.cpp

```
#include<stdio.h>
#include<string.h>
int i=1,j=0,no=0,tmpch=90;
char str[100],left[15],right[15];
void findopr();
void explore();
void fleft(int);
void fright(int);
struct exp
{
int pos;
char op;
}k[15];
void main()
{
printf("\t\tINTERMEDIATE CODE GENERATION OF DAG\n\n");
scanf("%s",str);
printf("The intermediate code:\t\tExpression\n");
findopr();
explore();
}
void findopr()
{
for(i=0;str[i]!='\0';i++)
 if(str[i]==':')
 {
 k[j].pos=i;
 k[j++].op=':';
 }
for(i=0;str[i]!='\0';i++)
 if(str[i]=='/')
 {
 k[j].pos=i;
 k[j++].op='/';
```

```
 }
for(i=0;str[i]!='\0';i++)
 if(str[i]=='*')
 {
 k[j].pos=i;
 k[j++].op='*';
 }
for(i=0;str[i]!='\0';i++)
 if(str[i]=='+')
 {
 k[j].pos=i;
 k[j++].op='+';
 }
for(i=0;str[i]!='\0';i++)
 if(str[i]=='-')
 {
 k[j].pos=i;
 k[j++].op='-';
 }
}
void explore()
{
i=1;
while(k[i].op!='\0')
{
 fleft(k[i].pos);
 fright(k[i].pos);
 str[k[i].pos]=tmpch--;
 printf("\t%c := %s%c%s\t\t",str[k[i].pos],left,k[i].op,right);
 for(j=0;j <strlen(str);j++)
 if(str[j]!='$')
 printf("%c",str[j]);
 printf("\n");
 i++;
```

```
}
fright(-1);
if(no==0)
{
 fleft(strlen(str));
 printf("\t%s := %s",right,left);
}
printf("\t%s := %c",right,str[k[--i].pos]);
}
void fleft(int x)
{
int w=0,flag=0;
x--;
while(x!= -1 &&str[x]!= '+'
&&str[x]!='*'&&str[x]!='='&&str[x]!='\0'&&str[x]!='-'&&str[x]!='/'&&str[x]!=':')
{
 if(str[x]!='$'&& flag==0)
 {
 left[w++]=str[x];
 left[w]='\0';
 str[x]='$';
 flag=1;
 }
 x--;
}
}
void fright(int x)
{
int w=0,flag=0;
x++;
while(x!= -1 && str[x]!=
'+'&&str[x]!='*'&&str[x]!='\0'&&str[x]!='='&&str[x]!=':'&&str[x]!='-
'&&str[x]!='/')
{
 if(str[x]!='$'&& flag==0)
 {
 right[w++]=str[x];
 right[w]='\0';
 str[x]='$';
 flag=1;
 }
 x++;
}
}
```

OUTPUT:



Result : The program Implementation of DAG was executed successsfully in online GDB compiler.

WEEK-13 Implementation of Storage Allocation Strategies(Stack)

Experiment 13

Date :- 11/04/22

Pranjay Poddar
RA1911028010129
CSE - CC J1

Implement any one storage allocation strategies

Aim :- To implement any of the stack storage allocation strategies.

Procedure :-

(1) Initially check whether the stack is empty
(2) Insert an element into the stack using push operation.
(3) Delete an element from the stack
(4) Display the elements.
(5) Top the stack element will be displayed

Calculation :-

Enter your choice (1-5) : 1
elements = 10
enter more elements : y

9/5 verified
11/4/22

PROGRAM:

main.c

```c
#include<stdio.h>
#include<stdlib.h>
#define TRUE 1
#define FALSE 0
typedef struct Heap
{
int data;
struct Heap *next;
}
node;
node *create();
void main()
{
int choice,val;
char ans;
node *head;
void display(node *);
node *search(node *,int);
node *insert(node *);
void dele(node **);
head=NULL;
do
{
printf("\nprogram to perform various operations on heap using dynamic memory
management");
printf("\n1.create");
printf("\n2.display");
printf("\n3.insert an element in a list");
printf("\n4.delete an element from list");
printf("\n5.quit");
printf("\nenter your chioce(1-5)");
scanf("%d",&choice);
switch(choice)
{
case 1:head=create();
break;
case 2:display(head);
break;
case 3:head=insert(head);
```

```
40  break;
41  case 4:dele(&head);
42  break;
43  case 5:exit(0);
44  default:
45  printf("invalid choice,try again");
46  }
47  }
48  while(choice!=5);
49  }
50  node* create()
51  {
52  node *temp,*New,*head;
53  int val,flag;
54  char ans='y';
55  node *get_node();
56  temp=NULL;
57  flag=TRUE;
58  do
59  {
60  printf("\n enter the element:");
61  scanf("%d",&val);
62  New=get_node();
63  if(New==NULL)
64  printf("\nmemory is not allocated");
65  New->data=val;
66  if(flag==TRUE)
67  {
68  head=New;
69  temp=head;
70  flag=FALSE;
71  }
72  else
73  {
74  temp->next=New;
75  temp=New;
76  }
77  printf("\ndo you want to enter more elements?(y/n)");
78  }
```

```
while(ans=='y');
printf("\nthe list is created\n");
return head;
}
node *get_node()
{
node *temp;
temp=(node*)malloc(sizeof(node));
temp->next=NULL;
return temp;
}
void display(node *head)
{
node *temp;
temp=head;
if(temp==NULL)
{
printf("\nthe list is empty\n");
return;
}
while(temp!=NULL)
{
printf("%d->",temp->data);
temp=temp->next;
}
printf("NULL");
}
node *search(node *head,int key)
{
node *temp;
int found;
temp=head;
if(temp==NULL)
{
printf("the linked list is empty\n");
return NULL;
}
found=FALSE;
while(temp!=NULL && found==FALSE)
```

```
{
if(temp->data!=key)
temp=temp->next;
else
found=TRUE;
}
if(found==TRUE)
{
printf("\nthe element is present in the list\n");
return temp;
}
else
{
printf("the element is not present in the list\n");
return NULL;
}
}
node *insert(node *head)
{
int choice;
node *insert_head(node *);
void insert_after(node *);
void insert_last(node *);
printf("n1.insert a node as a head node");
printf("n2.insert a node as a head node");
printf("n3.insert a node at intermediate position in t6he list");
printf("\nenter your choice for insertion of node:");
scanf("%d",&choice);
switch(choice)
{
case 1:head=insert_head(head);
break;
case 2:insert_last(head);
break;
case 3:insert_after(head);
break;
}
return head;
}
```

```
node *insert_head(node *head)
{
node *New,*temp;
New=get_node();
printf("\nEnter the element which you want to insert");
scanf("%d",&New->data);
if(head==NULL)
head=New;
else
{
temp=head;
New->next=temp;
head=New;
}
return head;
}
void insert_last(node *head)
{
node *New,*temp;
New=get_node();
printf("\nenter the element which you want to insert");
scanf("%d",&New->data);
if(head==NULL)
head=New;
else
{
temp=head;
while(temp->next!=NULL)
temp=temp->next;
temp->next=New;
New->next=NULL;
}
}
void insert_after(node *head)
{
int key;
node *New,*temp;
New=get_node();
printf("\nenter the elements which you want to insert");
```

```
196  scanf("%d",&New->data);
197  if(head==NULL)
198  {
199  head=New;
200  }
201  else
202  {
203  printf("\enter the element which you want to insert the node");
204  scanf("%d",&key);
205  temp=head;
206  do
207  {
208  if(temp->data==key)
209  {
210  New->next-temp->next;
211  temp->next=New;
212  return;
213  }
214  else
215  temp=temp->next;
216  }
217  while(temp!=NULL);
218  }
219  }
220  node *get_prev(node *head,int val)
221  {
222  node *temp,*prev;
223  int flag;
224  temp=head;
225  if(temp==NULL)
226  return NULL;
227  flag=FALSE;
228  prev=NULL;
229  while(temp!=NULL && ! flag)
230  {
231  if(temp->data!=val)
232  {
233  prev=temp;
234  temp=temp->next;
```

```
}
else
flag=TRUE;
}
if(flag)
return prev;
else
return NULL;
}
void dele(node **head)
{
node *temp,*prev;
int key;
temp=*head;
if(temp==NULL)
{
printf("\nthe list is empty\n");
return;
}
printf("\nenter the element you want to delete:");
scanf("%d",&key);
temp=search(*head,key);
if(temp!=NULL)
{
prev=get_prev(*head,key);
if(prev!=NULL)
{
prev->next=temp->next;
free(temp);
}
else
{
*head=temp->next;
free(temp);
}
printf("\nthe element is deleted\n");
}
}
```

OUTPUT:

```
program to perform various operations on heap using dynamic memory management
1.create
2.display
3.insert an element in a list
4.delete an element from list
5.quit
enter your chioce(1-5)3
n1.insert a node as a head noden2.insert a node as a head noden3.insert a node at intermediate position in t6he list
enter your choice for insertion of node:3

enter the elements which you want to insert1

program to perform various operations on heap using dynamic memory management
1.create
2.display
3.insert an element in a list
4.delete an element from list
5.quit
enter your chioce(1-5)
```

Result : The program Implementation of Storage Allocation Strategies (Stack) was executed successsfully in online GDB compiler.