

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: -

Sr. no.	Experiment	PAGE No.	DATE: -
1	Implementation of Lexical Analyser	3	11.1.22
2	Conversion from Regular Expression to NFA	8	28.1.22
3	Conversion from NFA to DFA	13	11.2.22
4	Elimination of Left Recursion and Left Factoring	17	18.2.22
5	Computation of FIRST AND FOLLOW	23	25.2.22
6	Construction of Predictive Parsing Table	30	04.3.22
7	Implementation of Shift Reduce Parsing	39	11.3.22
8	Computation of LEADING AND TRAILING	44	28.3.22
9	Computation of LR (0) items	52	28.3.22
10	Intermediate code generation – Postfix, Prefix	61	04.4.22
11	Implementation of Intermediate code generation Quadruple, Triple, Indirect triple	67	04.4.22
12	Implementation of DAG.	72	04.4.22
13	Implementation of storage allocation strategy.	78	04.4.22

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
CSE-CC J1

RA1911028010129

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 is then checks for the identifier
- 5) Stop the program

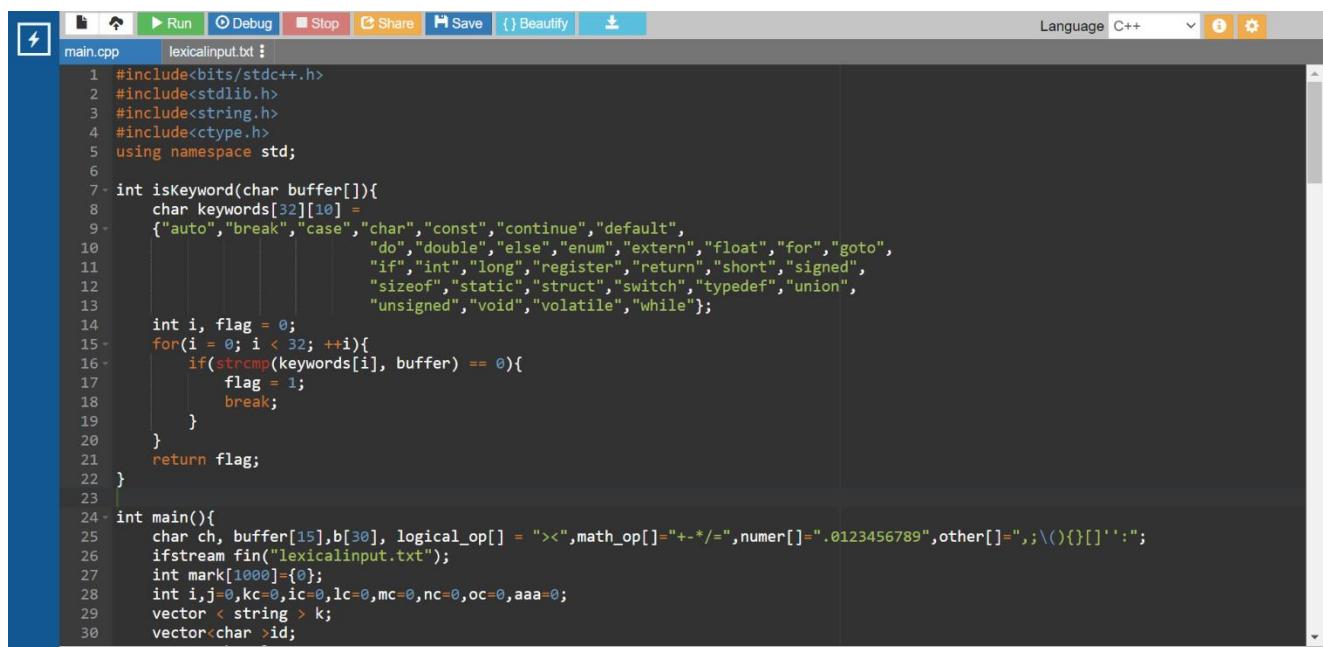
Procedure :-

- 1) taken generated is of the form <id, number>
- 2) If becomes here an operator then

cker gene related.

The streams of tokens generated are displayed in the console output step.

Code:



```
main.cpp lexicalinput.txt : Language C++ 
1 #include<bits/stdc++.h>
2 #include<stdlib.h>
3 #include<string.h>
4 #include<ctype.h>
5 using namespace std;
6
7 int isKeyword(char buffer[]){
8     char keywords[32][10] =
9     {"auto","break","case","char","const","continue","default",
10      "do","double","else","enum","extern","float","for","goto",
11      "if","int","long","register","return","short","signed",
12      "sizeof","static","struct","switch","typedef","union",
13      "unsigned","void","volatile","while"};
14     int i, flag = 0;
15     for(i = 0; i < 32; ++i){
16         if(strcmp(keywords[i], buffer) == 0){
17             flag = 1;
18             break;
19         }
20     }
21     return flag;
22 }
23
24 int main(){
25     char ch, buffer[15], b[30], logical_op[] = "><", math_op[] = "+-*=/", numer[] = ".0123456789", other[] = ";\\(){}[]`";
26     ifstream fin("lexicalinput.txt");
27     int mark[1000]={0};
28     int i,j=0,kc=0,ic=0,lc=0,mc=0,nc=0,oc=0,aaa=0;
29     vector<string> K;
30     vector<char> id;
```

```

main.cpp lexicalinput.txt : 
31     vector<char>lo;
32     vector<char>ma;
33     vector<string>nu;
34     vector<char>ot;
35     if(!fin.is_open()){
36         cout<<"error while opening the file\n";
37         exit(0);
38     }
39
40     while(!fin.eof()){
41         ch = fin.get();
42         for(i = 0; i < 12; ++i){
43             if(ch == other[i]){
44                 int aa=ch;
45                 if(mark[aa]!=1){
46                     ot.push_back(ch);
47                     mark[aa]=1;
48                     ++oc;
49                 }
50             }
51         }
52         for(i = 0; i < 5; ++i){
53             if(ch == math_op[i]){
54                 int aa=ch;
55                 if(mark[aa]!=1){
56                     ma.push_back(ch);
57                     mark[aa]=1;
58                     ++mc;
59                 }
60             }
61         }
62         for(i = 0; i < 2; ++i){
63             if(ch == logical_op[i]){
64                 int aa=ch;
65                 if(mark[aa]!=1){
66                     lo.push_back(ch);
67                     mark[aa]=1;
68                     ++lc;
69                 }
70             }
71         }
72         if(ch=='0' || ch=='1' || ch=='2' || ch=='3' || ch=='4' || ch=='5' || ch=='6' ||
73         ch=='7' || ch=='8' || ch=='9' || ch=='.')|| ch == ' ' || ch == '\n' || ch == ';'){
74             if(ch=='0' || ch=='1' || ch=='2' || ch=='3' || ch=='4' || ch=='5' || ch=='6' ||
75             || ch=='7' || ch=='8' || ch=='9' || ch=='.')b[aaa++]=ch;
76             if((ch == ' ' || ch == '\n' || ch == ';') && (aaa != 0)){
77                 b[aaa] = '\0';
78                 aaa = 0;
79                 char arr[30];
80                 strcpy(arr,b);
81                 nu.push_back(arr);
82                 ++nc;
83             }
84         }
85         if(isalnum(ch)){
86             buffer[j++] = ch;
87         }
88         else if((ch == ' ' || ch == '\n') && (j != 0)){
89             buffer[j] = '\0';
90             j = 0;

```

```
main.cpp lexicalinput.txt
91     if(isKeyword(buffer) == 1){
92         k.push_back(buffer);
93         ++kc;
94     }
95     else{
96         if(buffer[0]>=97 && buffer[0]<=122) {
97             if(mark[buffer[0]-'a']+1){
98                 id.push_back(buffer[0]);
99                 ++ic;
100                mark[buffer[0]-'a']=1;
101            }
102        }
103    }
104 }
105 fin.close();
106 printf("Keywords: ");
107 for(int f=0;f<kc;++f){
108     if(f==kc-1){
109         cout<<k[f]<<"\n";
110     }
111     else {
112         cout<<k[f]<<, ";
113     }
114 }
115 printf("Identifiers: ");
116 for(int f=0;f<ic;++f){
117     if(f==ic-1){
118         cout<<id[f]<<"\n";
119     }
120 }
```

```
main.cpp lexicalinput.txt
121     else {
122         cout<<id[f]<<, ";
123     }
124 }
125 printf("Math Operators: ");
126 for(int f=0;f<mc;++f){
127     if(f==mc-1){
128         cout<<ma[f]<<"\n";
129     }
130     else {
131         cout<<ma[f]<<, ";
132     }
133 }
134 printf("Logical Operators: ");
135 for(int f=0;f<lci;++f){
136     if(f==lci-1){
137         cout<<lo[f]<<"\n";
138     }
139     else {
140         cout<<lo[f]<<, ";
141     }
142 }
143 printf("Numerical Values: ");
144 for(int f=0;f<nc;++f){
145     if(f==nc-1){
146         cout<<nu[f]<<"\n";
147     }
148     else {
149         cout<<nu[f]<<, ";
150     }
}
```

```
133 }
134 printf("Logical Operators: ");
135 for(int f=0;f<lcl;f++){
136     if(f==lcl-1){
137         cout<<lo[f]<<"\n";
138     }
139     else {
140         cout<<lo[f]<<, " ;
141     }
142 }
143 printf("Numerical Values: ");
144 for(int f=0;f<nc;f++){
145     if(f==nc-1){
146         cout<<nu[f]<<"\n";
147     }
148     else {
149         cout<<nu[f]<<, " ;
150     }
151 }
152 printf("Others: ");
153 for(int f=0;f<oc;f++){
154     if(f==oc-1){
155         cout<<ot[f]<<"\n";
156     }
157     else {
158         cout<<ot[f]<<, " ;
159     }
160 }
161 return 0;
162 }
```

Input:

```
main.cpp | lexicalinput.txt |
1 int z = 10;
2 float q = 53.5;
3 if(q > 10){
4 char name[100] = "Hello my name is Pranjay";
5 }
```

Output:

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

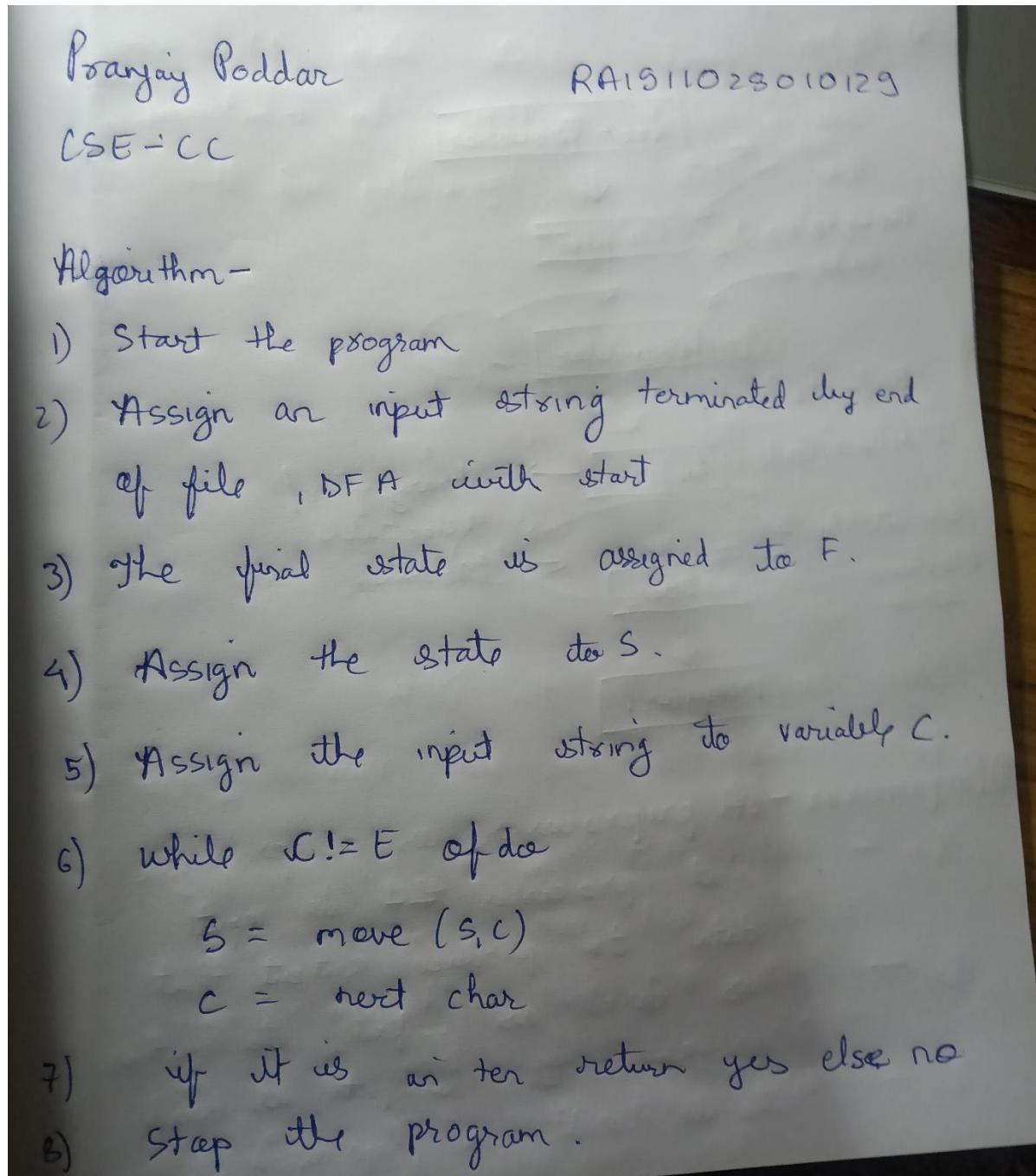
...Program finished with exit code 0
Press ENTER to exit console.[]
```

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

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

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

Algorithm:



Procedure -

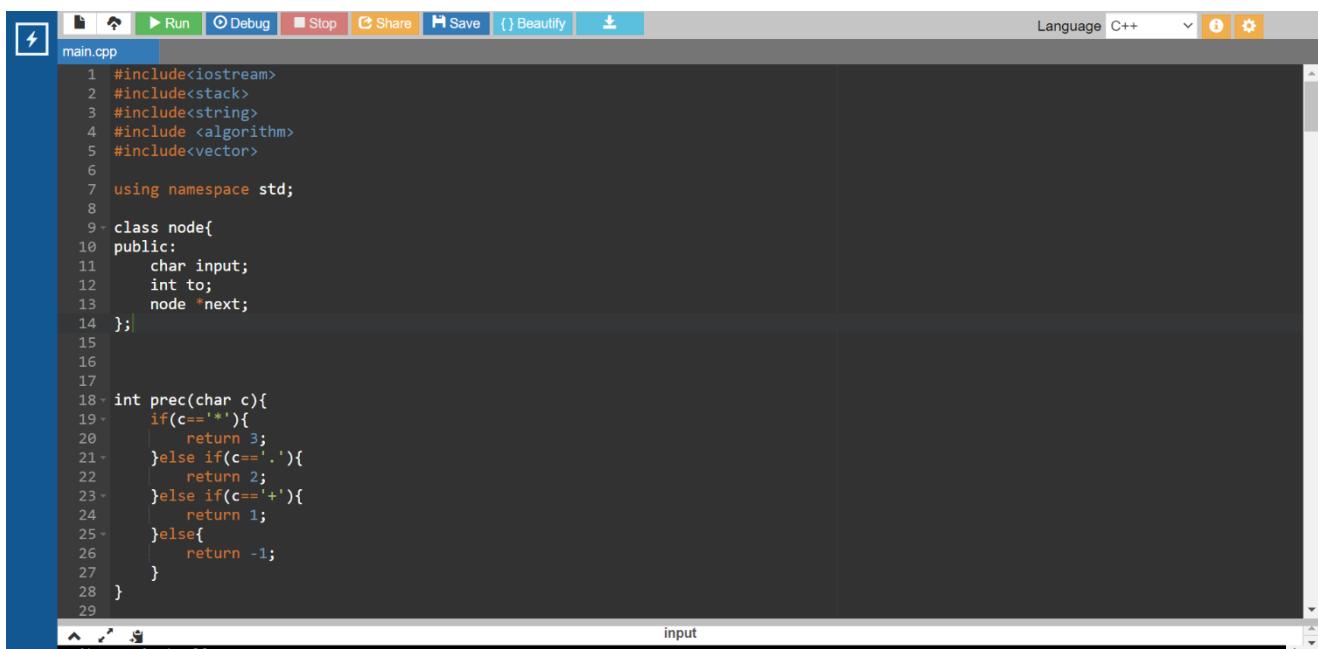
Step 1 - A structure is created for each kind of regular expression the user want.

Step 2 - Under each structure, we define 3 variables.

Step 3 - Each state changes.

Step 4 - A switch case is made with 9 choices and based on the choice select an NFA.

Code:



The screenshot shows a code editor window with the following details:

- File Name:** main.cpp
- Toolbar:** Includes icons for Run, Debug, Stop, Share, Save, Beautify, and a build status icon.
- Language:** C++
- Code Content:**

```
1 #include<iostream>
2 #include<stack>
3 #include<string>
4 #include <algorithm>
5 #include<vector>
6
7 using namespace std;
8
9 class node{
10 public:
11     char input;
12     int to;
13     node *next;
14 };
15
16
17
18 int prec(char c){
19     if(c=='*'){
20         return 3;
21     }else if(c==' .'){
22         return 2;
23     }else if(c=='+' ){
24         return 1;
25     }else{
26         return -1;
27     }
28 }
```
- Status Bar:** Shows "input" at the bottom right.

```

main.cpp
32 string post(string s)
33 {
34     stack<char> st;
35     st.push('N');
36     int l = s.length();
37     string ns;
38     for(int i = 0; i < l; i++)
39     {
40         if((s[i] >= 'a' && s[i] <= 'z')||(s[i] >= 'A' && s[i] <= 'Z')){
41             ns+=s[i];
42         }
43
44         else if(s[i] == '('){
45             st.push('(');
46         }
47         else if(s[i] == ')')
48         {
49             while(st.top() != 'N' && st.top() != '(')
50             {
51                 char c = st.top();
52                 st.pop();
53                 ns += c;
54             }
55             if(st.top() == '(')
56             {
57                 char c = st.top();
58                 st.pop();
59             }
60         }
61     }
62     char c = st.top();
63     st.pop();
64 }
65 else{
66     while(st.top() != 'N' && prec(s[i]) <= prec(st.top()))
67     {
68         char c = st.top();
69         st.pop();
70         ns += c;
71     }
72     st.push(s[i]);
73 }
74 while(st.top() != 'N')
75 {
76     char c = st.top();
77     st.pop();
78     ns += c;
79 }
80 return ns;
81 }
82 void printnode(vector<node*> v){
83     cout<<"| from state\t| input\t| tostates"<<endl;
84     cout<<"| " <<i<<"\t| " <<v[i]<<"\t| ";
85     for(int i=0;i<v.size();i++){
86         cout<<"| " <<i<<"\t| " <<v[i]<<"\t| ";
87     }
88 }
89 cout<<"| from state\t| input\t| tostates"<<endl;
90 for(int i=0;i<v.size();i++){
91     cout<<"| " <<i<<"\t| " <<v[i]<<"\t| ";
92     node* head = v[i];
93     cout<<head->input;
94     bool first = true;
95     while(head!=NULL){
96         if (first)
97         {
98             cout<<"\t| " <<head->input<<"\t| ";
99             first = false;
100        }
101        cout<<head->to;
102        head = head->next;
103    }
104 }
105 node* makenode(char in){
106     node* a = new node;
107     a->input = in;
108     a->to = -1;
109     a->next = NULL;
110 }

```

```

main.cpp
113     node* copynode(node *n){
114         node* b = new node;
115         b->input = -1;
116         b->to = -1;
117         b->next =NULL;
118         return b;
119     }
120
121     void addd(vector<node*> &v,vector<vector<int> > &st){
122         int x,y;
123         int first,last1;
124         y = st[st.size()-1][0];
125         x = st[st.size()-2][1];
126         first = st[st.size()-2][0];
127         last1 = st[st.size()-1][1];
128
129         st.pop_back();
130         st.pop_back();
131
132         vector<int> ptemp;
133         ptemp.push_back(first);
134         ptemp.push_back(last1);
135         st.push_back(ptemp);
136
137         node* last = v[y];
138         node * lnode= v[x];
139         node* temp = copynode(last);
140         // temp->to = -1;
141         while(lnode->next!=NULL)if
142
143     }
144
145     void orr(vector<node*> &v,vector<vector<int> > &st){
146         int x,y,x1,y1;
147         x = st[st.size()-2][0];
148         y = st[st.size()-1][0];
149         x1 = st[st.size()-2][1];
150         y1 = st[st.size()-1][1];
151         node* start = makenode('e');
152         node* end = makenode('e');
153         v.push_back(start);
154         int firstnode = v.size() -1;
155         v.push_back(end);
156         int endnode = v.size() -1;
157
158         st.pop_back();
159         st.pop_back();
160
161         vector<int> ptemp;
162         ptemp.push_back(firstnode);
163         ptemp.push_back(endnode);
164         st.push_back(ptemp);
165
166         for(int i=0;i<v.size()-2;i++){
167             node* h=v[i];
168             while(h->next!=NULL){
169                 if(h->to==x || h->to == y){
170                     h->to = firstnode;
171                 }
172             }
173         }
174
175         node* adlink = v[x1];
176         while(adlink->next!=NULL){
177             adlink = adlink->next;
178         }
179
180         adlink->to= endnode;
181         adlink->next = copynode(end);
182
183         node* adlink1 = v[y1];
184         while(adlink1->next!=NULL){
185             adlink1 = adlink1->next;
186         }
187         adlink1->to = endnode;
188         adlink1->next = copynode(end);
189
190     }
191
192     void closure(vector<node*> &v, vector<vector<int> > &st){
193         int x,x1;
194         x = st[st.size()-1][0];
195         x1 = st[st.size()-1][1];
196         node* s = makenode('e');
197         // node* e = makenode('e');
198         v.push_back(s);
199         int firstnode = v.size() -1;
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218

```

```

main.cpp
271     v.push_back(ptemp);
272     vector<int> ptemp;
273     ptemp.push_back(v.size()-1);
274     ptemp.push_back(v.size()-1);
275     st.push_back(ptemp);
276 }
277 else if(o[1]=='.'){
278     andd(v,st);
279 }
280 else if(o[1]=='+'){
281     orr(v,st);
282 }
283 else if(o[1]=='*'){
284     closure(v,st);
285 }
286
287 }
288 cout<<"\ntransition table for given regular expression is - \n";
289 printnode(v);
290 cout<<endl;
291 cout<<"starting node is ";
292 cout<<st[st.size()-1][0]<<endl;
293 cout<<"ending node is ";
294 cout<<st[st.size()-1][1]<<endl;
295 return 0;
296 }
297 }
```

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_\.]+\.)+ @()

transitions 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 | -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 - 12/02/2022

Pranay 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 $a \in \Sigma$ 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 $s \in S$ is a final state in the NFA then mark S as a final state in the DFA.

PROGRAM:

The screenshot shows a C++ IDE interface with the following details:

- Toolbar:** Includes icons for Run, Debug, Stop, Share, Save, Beautify, and a download arrow.
- Language:** Set to C++.
- Code Editor:** The main window displays the source code for a C++ program named `main.cpp`. The code is a solution for converting a Non-deterministic Finite Automaton (NFA) to a Deterministic Finite Automaton (DFA). It uses vectors of vectors to represent the states and transitions of both automata. The code includes logic to calculate transitions based on input symbols and to handle state minimization or conversion rules. The code is annotated with line numbers (e.g., 1, 2, 3, ..., 72) on the left side.

```

73         cout<<"\n"<<"dfa[{"<<dstate[k]/10 << ", " << dstate[k]%10 <<"}, a]: {"<<dfa[k][1]/10 << ", " << dfa[k][1]%10 << "}";
74     }
75     else{
76         cout<<"\n"<<"dfa[{"<<dstate[k]/10 << ", " << dstate[k]%10 <<"}, a]: "<<dfa[k][1];
77     }
78 }
79 else{
80     if (dfa[k][1] > 10){
81         cout<<"\n"<<"dfa["<<dstate[k] << ", a]: {"<<dfa[k][1]/10 << ", " << dfa[k][1]%10 << "}";
82     }
83     else{
84         cout<<"\n"<<"dfa["<<dstate[k] << ", a]: "<<dfa[k][1];
85     }
86 }
87 if (dstate[k] > 10){
88     if (dfa[k][2] > 10){
89         cout<<"\n"<<"dfa[{"<<dstate[k]/10 << ", " << dstate[k]%10 <<"}, b]: {"<<dfa[k][2]/10 << ", " << dfa[k][2]%10 << "}";
90     }
91     else{
92         cout<<"\n"<<"dfa[{"<<dstate[k]/10 << ", " << dstate[k]%10 <<"}, b]: "<<dfa[k][2];
93     }
94 }
95 else{
96     if (dfa[k][1] > 10){
97         cout<<"\n"<<"dfa["<<dstate[k] << ", b]: {"<<dfa[k][2]/10 << ", " << dfa[k][2]%10 << "}";
98     }
99     else{
100        cout<<"\n"<<"dfa["<<dstate[k] << ", b]: "<<dfa[k][2];
101    }
102 }
103 flag=0;
104 for(j=1;j<n;j++)
105 {
106     if(dfa[k][1]!=dstate[j])
107         flag++;
108 }

```

```

109     if(flag==n-1)
110     {
111         dstate[i++]=dfa[k][1];
112         n++;
113     }
114     flag=0;
115     for(j=1;j<n;j++)
116     {
117         if(dfa[k][2]!=dstate[j])
118             flag++;
119     }
120     if(flag==n-1)
121     {
122         dstate[i++]=dfa[k][2];
123         n++;
124     }
125     k++;
126 }
127 return 0;
128 }
129

```

OUTPUT:

The screenshot shows a terminal window within an online GDB compiler. The terminal has a light gray header bar with various icons and buttons: a file icon, a cloud icon, a green 'Run' button, a blue 'Debug' button, a red 'Stop' button, an orange 'Share' button, a blue 'Save' button, a 'Beautify' button, and a download icon. To the right of the header is the word 'input'. The main area of the terminal displays the following text:

```
nfa [1, a]: 12
nfa [1, b]: 1
nfa [2, a]: 0
nfa [2, b]: 3
nfa [3, a]: 0
nfa [3, b]: 4
nfa [4, a]: 0
nfa [4, b]: 0

dfa[1, a]: {1, 2}
dfa[1, b]: 1
dfa[{1, 2}, a]: {1, 2}
dfa[{1, 2}, b]: {1, 3}
dfa[{1, 3}, a]: {1, 2}
dfa[{1, 3}, b]: {1, 4}
dfa[{1, 4}, a]: {1, 2}

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

Result : The program to convert NFA to DFA was executed successfully 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 separate the production v start working.

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

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

$$S \rightarrow CS' | -S'$$

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

$$S' \rightarrow z | as' | b'$$

(ii) To remove left factoring

- For the corner prefers we must ~~only~~ L production
- So here the corner prefers 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 reversion & then left factoring is known as left factor grammar.

PROGRAM:

The screenshot shows a web-based C++ compiler interface. The URL in the address bar is [onlinegdb.com/online_c++_compiler](https://onlinegdb.com/online_c%2B%2B_compiler). The main window displays the code for `main.cpp`. The code implements a parser for a grammar where a non-terminal can have multiple productions separated by '|'. It reads the parent non-terminal from input, reads the total number of productions, and then reads each production into a string `a`, concatenating them with '|'. Finally, it prints the resulting string `a`.

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4 int main()
5 {
6     int n, j, l, i, m;
7     int len[10] = {};
8     string a, b1, b2, flag;
9     char c;
10    cout << "Enter the Parent Non-Terminal : ";
11    cin >> c;
12    a.push_back(c);
13    b1 += a + "\' ->";
14    b2 += a + "\'\' ->";
15    ;
16    a += "->";
17    cout << "Enter total number of productions : ";
18    cin >> n;
19    for (i = 0; i < n; i++)
20    {
21        cout << "Enter the Production " << i + 1 << " : ";
22        cin >> flag;
23        len[i] = flag.size();
24        a += flag;
25        if (i != n - 1)
26        {
27            a += "|";
28        }
29    }
30    cout << "The Production Rule is : " << a << endl;
```

```

31     char x = a[3];
32     for (i = 0, m = 3; i < n; i++)
33     {
34         if (x != a[m])
35         {
36             while (a[m++] != '|')
37                 ;
38         }
39         else
40         {
41             if (a[m + 1] != '|')
42             {
43                 b1 += "|" + a.substr(m + 1, len[i] - 1);
44                 a.erase(m - 1, len[i] + 1);
45             }
46             else
47             {
48                 b1 += "#";
49                 a.insert(m + 1, 1, a[0]);
50                 a.insert(m + 2, 1, '\\'');
51                 m += 4;
52             }
53         }
54     }
55     char y = b1[6];
56     for (i = 0, m = 6; i < n - 1; i++)
57     {
58         if (y == b1[m])
59         {
60             if (b1[m + 1] != '|')

```

```
60             if (b1[m + 1] != '|')
61             {
62                 flag.clear();
63                 for (int s = m + 1; s < b1.length(); s++)
64                 {
65                     flag.push_back(b1[s]);
66                 }
67                 b2 += "|" + flag;
68                 b1.erase(m - 1, flag.length() + 2);
69             }
70         else
71         {
72             b1.insert(m + 1, 1, b1[0]);
73             b1.insert(m + 2, 2, '\n');
74             b2 += "#";
75             m += 5;
76         }
77     }
78 }
79 b2.erase(b2.size() - 1);
80 cout << "After Left Factoring : " << endl;
81 cout << a << endl;
82 cout << b1 << endl;
83 cout << b2 << endl;
84 return 0;
85 }
```

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 successfully in online GDB compiler.

WEEK-5 Computation of FIRST AND FOLLOW

Exp No:- 05

Date: 24/02/2022

Pranjay Poddar
RA1911028010129
CSE-CC J1Computation of First & Follow

Aim :- To compute $\text{First}(x)$ and $\text{Follow}(x)$ where x is a grammar symbol.

Procedure :-

(i) For obtaining First :-

- If x is a terminal, then $\text{First}(x) = \{x\}$
- If $x \rightarrow \epsilon$ is a production, then add ϵ to $\text{First}(x)$.
- If x is a non-terminal and $x \rightarrow y_1, y_2, \dots, y_k$ is a production, then add $\text{First}(y_1)$ to $\text{First}(x)$. If y_i derives ϵ , then add $\text{First}(y_i)$ to $\text{First}(x)$.

(ii) For obtaining Follow -

- The follow (start symbol) place, $\$$, where $\$$ is the input end marker.
- If there is a production $A \rightarrow dB\beta$, then everything in $\text{First}(\beta)$ except ϵ is in $\text{Follow}(A)$.
- If there is a production $A \rightarrow aB$ or a production $A \rightarrow aB\beta$ where $\text{First}(\beta)$

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
$S \rightarrow Bb \mid cd$	$\{\epsilon, a, b, c, d\}$	$\{\epsilon, \$\}$
$B \rightarrow ab \mid \epsilon$	$\{a, \epsilon\}$	$\{b\}$
$C \rightarrow cc \mid \epsilon$	$\{c, \epsilon\}$	$\{d\}$

Following the above steps we obtain this.

PROGRAM:

The screenshot shows a web-based C++ editor interface. At the top, there's a navigation bar with a back arrow, a refresh icon, a URL field containing "onlinetgdb.com/edit/8XyZXfayd", and several buttons: Run (green), Debug (blue), Stop (red), Share (orange), Save (blue), Beautify (light blue), and a download icon.

The main area displays the code for "main.cpp". The code implements a Depth-First Search (DFS) algorithm to find First and Follow sets for a grammar. It includes a function "dfs" that takes parameters: a character index "i", an origin character "org", a last character "last", a map of characters to vectors of characters "mp", and a set "ss" to store results. The "dfs" function iterates over rules "r" in "mp[i]". For each rule, it checks if the current character "s" is the start symbol "i" or an end symbol 'e'. If so, it inserts "s" into the set and returns true. Otherwise, it checks if "s" is between 'A' and 'Z' and not equal to 'e'. If true, it inserts "s" into the set and continues to the next character in the rule. If false, it checks if the origin "org" is the same as the current character "i" or if "i" is the last character. If true, it inserts "s" into the set and marks "rtake" as true. Finally, it calls "dfs" recursively for the next character in the rule, passing the current character as the origin. The "main" function initializes variables "i" and "j" and calls the "dfs" function.

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 set<char> ss;
4 bool dfs(char i, char org, char last, map<char,vector<vector<char>>> &mp){
5     bool rtake = false;
6     for(auto r : mp[i]){
7         bool take = true;
8         for(auto s : r){
9             if(s == i) break;
10            if(!take) break;
11            if(!(s>='A'&&s<='Z')&&s != 'e'){
12                ss.insert(s);
13                break;
14            }
15            else if(s == 'e'){
16                if(org == i||i == last)
17                    ss.insert(s);
18                rtake = true;
19                break;
20            }
21            else{
22                take = dfs(s,org,r[r.size()-1],mp);
23                rtake |= take;
24            }
25        }
26    }
27    return rtake;
28 }
29 int main(){
30     int i,j;
```

→ C onlinegdb.com/edit/8XyZXfayd

File Run Debug Stop Share Save Beautify

main.cpp inputfirstfollow.txt

```
29 int main(){
30     int i,j;
31     ifstream fin("inputfirstfollow.txt");
32     string num;
33     vector<int> fs;
34     vector<vector<int>> a;
35     map<char,vector<vector<char>>> mp;
36     char start;
37     bool flag = 0;
38     cout<<"Grammar: "<<' \n';
39     while(getline(fin,num)){
40         if(flag == 0) start = num[0],flag = 1;
41         cout<<num<<' \n';
42         vector<char> temp;
43         char s = num[0];
44         for(i=3;i<num.size();i++){
45             if(num[i] == '|'){
46                 mp[s].push_back(temp);
47                 temp.clear();
48             }
49             else temp.push_back(num[i]);
50         }
51         mp[s].push_back(temp);
52     }
53     map<char,set<char>> fmp;
54     for(auto q : mp){
55         ss.clear();
56         dfs(q.first,q.first,q.first,mp);
57         for(auto g : ss) fmp[q.first].insert(g);
58     }
```

The screenshot shows a code editor interface with a dark theme. At the top, there is a toolbar with icons for file operations (New, Open, Save, etc.) and project management (Run, Debug, Stop, Share, Save, Beautify). Below the toolbar, the current file is listed as "main.cpp" and the active tab is "inputfirstfollow.txt".

The code itself is a C++ program. It starts by defining a function that prints the first set for each non-terminal symbol in the grammar. This function uses a map `fmp` where keys are non-terminals and values are sets of terminals. It iterates over each entry in `fmp`, concatenating the first terminal of each entry into a string `ans` and then adding the remaining terminals from the second set of each entry. Finally, it prints `ans` followed by a newline.

After this, a global map `gmp` is initialized with a single entry for the start symbol containing the dollar sign (`'\$'`). A loop then runs 10 times, performing the following steps:

- It iterates over each entry in `mp` (which contains terminals).
- For each terminal, it iterates over its second set.
- For each entry in the second set, it iterates over the characters from index 0 to size minus 1.
- If the character is between 'A' and 'Z', it checks if the next character is also between 'A' and 'Z'. If so, it adds the character to the global map `gmp` for the current terminal. If not, it enters a loop where it continues to add characters until it finds one that is not between 'A' and 'Z'.
- If the character is not between 'A' and 'Z', it enters a loop where it continues to add characters until it finds one that is between 'A' and 'Z'.
- If the character found in the previous step is 'e', it iterates over the entries in `fmp` for that character and adds them to the global map `gmp`.

```
58     }
59     cout<<'\n';
60     cout<<"FIRST: "<<'\n';
61     for(auto q : fmp){
62         string ans = "";
63         ans += q.first;
64         ans += " = {";
65         for(char r : q.second){
66             ans += r;
67             ans += ',';
68         }
69         ans.pop_back();
70         ans+="}";
71         cout<<ans<<'\n';
72     }
73     map<char, set<char>> gmp;
74     gmp[start].insert('$');
75     int count = 10;
76     while(count--){
77         for(auto q : mp){
78             for(auto r : q.second){
79                 for(i=0;i<r.size()-1;i++){
80                     if(r[i]>='A'&&r[i]<='Z'){
81                         if(!(r[i+1]>='A'&&r[i+1]<='Z')) gmp[r[i]].insert(r[i+1]);
82                         else {
83                             char temp = r[i+1];
84                             int j = i+1;
85                             while(temp>='A'&&temp<='Z'){
86                                 if(*fmp[temp].begin()=='e'){
87                                     for(auto g : fmp[temp]){
88                                         gmp[r[i]].insert(g);
89                                     }
90                                 }
91                             }
92                         }
93                     }
94                 }
95             }
96         }
97     }
98 }
```



INPUT:

The screenshot shows a code editor window with a dark theme. At the top, there is a toolbar with icons for file operations (New, Open, Save, Run, Debug, Stop). Below the toolbar, the file name "main.cpp" is displayed. The code area contains three numbered production rules:

- 1 $S \rightarrow Bb \mid Cd$
- 2 $B \rightarrow aB \mid e$
- 3 $C \rightarrow cC \mid e$

OUTPUT:

The terminal window displays the following output:

```
Grammar:  
S->Bb | Cd  
B->aB | e  
C->cC | e  
  
FIRST:  
B = {a, e}  
C = {c, e}  
S = {a, b, c, d}  
  
FOLLOW:  
B = {b}  
C = {d}  
S = {$}  
  
... Program finished with exit code 0  
Press ENTER to exit console.
```

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

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

Experiment-6

Date - 4/3/2022

Pranjay Poddar
RA1911028010129
J, CSE-CCConstruction Of Predictive Parsing Table

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

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

for ~~E~~ $E \rightarrow TE'$
 . $E' \rightarrow +TE' | E$
 T $\rightarrow FT'$
 T $\rightarrow *FT' | E$
 F $\rightarrow id | E$

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

Referring to the above example -

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

- (3) For each production if $A \rightarrow \alpha$ then,
- (1) Find first (α) and for each terminal in first (α), we make an entry $A \rightarrow \alpha$ in the table.
 - (2) If first (α) contains ϵ as a terminal then find follow (A) and make an entry in the table A.
 - (3) If the first (α) contains ϵ and follow (A) is a terminal, then make entry $A \rightarrow \alpha$ in the forst.

for the example taken \rightarrow

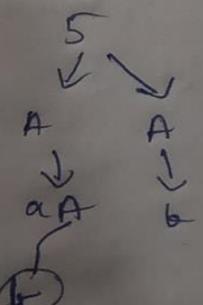
E	$\overset{id}{E \rightarrow TE'}$	$+$	$*$	C	$)$	S
E'				$E \rightarrow TE'$	$E^L \rightarrow E$	$E \overset{L}{\rightarrow} E$
T	$T \rightarrow FT'$	$F \rightarrow TF'$		$T * T'$		
T'		$T' \rightarrow E$	$T' \rightarrow FT'$		$T' \rightarrow B$	$T' \rightarrow E$
F	$F \rightarrow id$			$T \rightarrow (S)$		

Manual Calculation of Lab Exercises \rightarrow

$$S \rightarrow AA \quad \text{--- } ①$$

$$A \rightarrow ab/b \quad \text{--- } A \rightarrow aA \quad \text{--- } ②$$

$$A \rightarrow b \quad \text{--- } ③$$

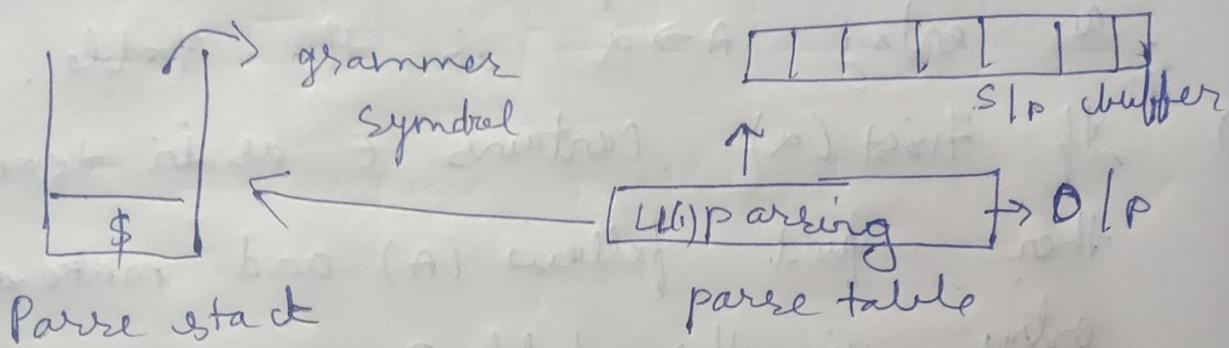


$$\text{First}(S) = \{a, b, \emptyset\}$$

$$\text{First}(A) = \{a, b\}$$

$\text{Follow}(S) = \{\$\}$ [follow of apart as \$]

$\text{Follow}(A) = \{a, t, \$\}$



String accepted = abbba

String rejected = abab

a	a	b	\$
S	1	1	
A	2	2	

PROGRAM:

The screenshot shows a web-based C editor interface. At the top, there's a navigation bar with icons for back, forward, and search, followed by the URL "onlinergdb.com/edit/iJEvx24tr". Below the URL is a toolbar with buttons for File, New, Run, Debug, Stop, Share, Save, and Beautify. The main area is a code editor with a dark background. The file name is "main.c" and the input file is "input.txt". The code itself is a C program with various comments explaining its purpose. It includes declarations for arrays like table, terminal, nonterminal, str, and first/follow, along with structures for product and production, and variables for no_pro and first_rhs.

```
1 #include<stdio.h>
2 #include<string.h>
3 #define TSIZE 128
4 // table[i][j] stores
5 // the index of production that must be applied on
6 // ith variable if the input is
7 // jth nonterminal
8 int table[100][TSIZE];
9 // stores all list of terminals
10 // the ASCII value if used to index terminals
11 // terminal[i] = 1 means the character with
12 // ASCII value is a terminal
13 char terminal[TSIZE];
14 // stores all list of terminals
15 // only Upper case letters from 'A' to 'Z'
16 // can be nonterminals
17 // nonterminal[i] means ith alphabet is present as
18 // nonterminal in the grammar
19 char nonterminal[26];
20 // structure to hold each production
21 // str[] stores the production
22 // len is the length of production
23 struct product {
24     char str[100];
25     int len;
26 }pro[20];
27 // no of productions in form A->β
28 int no_pro;
29 char first[26][TSIZE];
30 char follow[26][TSIZE];
31 // stores first of each production in form A->β
32 char first_rhs[100][TSIZE];
33 // check if the symbol is nonterminal
```

```

33 // check if the symbol is nonterminal
34 int isNT(char c) {
35     return c >= 'A' && c <= 'Z';
36 }
37 // reading data from the file
38 void readFromFile() {
39     FILE* fptr;
40     fptr = fopen("input.txt", "r");
41     char buffer[255];
42     int i;
43     int j;
44     while (fgets(buffer, sizeof(buffer), fptr)) {
45         printf("%s", buffer);
46         j = 0;
47         nonterminal[buffer[0] - 'A'] = 1;
48         for (i = 0; i < strlen(buffer) - 1; ++i) {
49             if (buffer[i] == '|') {
50                 ++no_pro;
51                 pro[no_pro - 1].str[j] = '\0';
52                 pro[no_pro - 1].len = j;
53                 pro[no_pro].str[0] = pro[no_pro - 1].str[0];
54                 pro[no_pro].str[1] = pro[no_pro - 1].str[1];
55                 pro[no_pro].str[2] = pro[no_pro - 1].str[2];
56                 j = 3;
57             }
58             else {
59                 pro[no_pro].str[j] = buffer[i];
60                 ++j;
61                 if (!isNT(buffer[i]) && buffer[i] != '-' && buffer[i] != '>') {
62                     terminal[buffer[i]] = 1;
63                 }
64             }
65         }
66         pro[no_pro].len = j;
67         ++no_pro;
68     }
69 }
70 void add_FIRST_A_to_FOLLOW_B(char A, char B) {
71     int i;
72     for (i = 0; i < TSIZE; ++i) {
73         if (i != '^')
74             follow[B - 'A'][i] = follow[B - 'A'][i] || first[A - 'A'][i];
75     }
76 }
77 void add_FOLLOW_A_to_FOLLOW_B(char A, char B) {
78     int i;
79     for (i = 0; i < TSIZE; ++i) {
80         if (i != '^')
81             follow[B - 'A'][i] = follow[B - 'A'][i] || follow[A - 'A'][i];
82     }
83 }
84 void FOLLOW() {
85     int t = 0;
86     int i, j, k, x;
87     while (t++ < no_pro) {
88         for (k = 0; k < 26; ++k) {
89             if (!nonterminal[k]) continue;
90             char nt = k + 'A';
91             for (i = 0; i < no_pro; ++i) {
92                 for (j = 3; j < pro[i].len; ++j) {
93                     if (nt == pro[i].str[j]) {
94                         for (x = j + 1; x < pro[i].len; ++x) {
95                             char sc = pro[i].str[x];
96                             if (isNT(sc)) {
97                                 add_FIRST_A_to_FOLLOW_B(sc, nt);
98                                 if (first[sc - 'A']['^'])
99                                     continue;
100                            }
101                        }
102                    }
103                }
104            }
105        }
106    }

```

```

101
102
103
104
105
106
107
108
109
110
111
112
113 }
114 void add_FIRST_A_to_FIRST_B(char A, char B) {
115     int i;
116     for (i = 0; i < TSIZE; ++i) {
117         if (i != '^') {
118             first[B - 'A'][i] = first[A - 'A'][i] || first[B - 'A'][i];
119         }
120     }
121 }
122 void FIRST() {
123     int i, j;
124     int t = 0;
125     while (t < no_pro) {
126         for (i = 0; i < no_pro; ++i) {
127             for (j = 0; j < pro[i].len; ++j) {
128                 char sc = pro[i].str[j];
129                 if (isNT(sc)) {
130                     add_FIRST_A_to_FIRST_B(sc, pro[i].str[0]);
131                     if (first[sc - 'A'][ '^'])
132                         continue;
133                 }
134             } else {
135                 first[pro[i].str[0] - 'A'][sc] = 1;
136             }
137             break;
138         }
139         if (j == pro[i].len)
140             first[pro[i].str[0] - 'A'][ '^'] = 1;
141     }
142     ++t;
143 }
144 }
145 void add_FIRST_A_to_FIRST_RHS_B(char A, int B) {
146     int i;
147     for (i = 0; i < TSIZE; ++i) {
148         if (i != '^')
149             first_rhs[B][i] = first[A - 'A'][i] || first_rhs[B][i];
150     }
151 }
152 // Calculates FIRST( $\beta$ ) for each  $A \rightarrow \beta$ 
153 void FIRST_RHS() {
154     int i, j;
155     int t = 0;
156     while (t < no_pro) {
157         for (i = 0; i < no_pro; ++i) {
158             for (j = 0; j < pro[i].len; ++j) {
159                 char sc = pro[i].str[j];
160                 if (isNT(sc)) {
161                     add_FIRST_A_to_FIRST_RHS_B(sc, i);
162                     if (first[sc - 'A'][ '^'])
163                         continue;
164                 }
165             } else {
166                 first_rhs[i][sc] = 1;
167             }
168             break;
169         }
170         ++t;
171     }
172 }

```

```

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  $\beta$ 
214     // in form A-> $\beta$ 
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

```

```

237     // printing parse table
238     printf("\n");
239     printf("\n\t*****LL(1) PARSING TABLE *****\n");
240     printf("\t-----\n");
241     printf("%-10s", "");
242     for (i = 0; i < TSIZE; ++i) {
243         if (terminal[i]) printf("%-10c", i);
244     }
245     printf("\n");
246     int p = 0;
247     for (i = 0; i < no_pro; ++i) {
248         if (i != 0 && (pro[i].str[0] != pro[i - 1].str[0]))
249             p = p + 1;
250         for (j = 0; j < TSIZE; ++j) {
251             if (first_rhs[i][j] && j != '^') {
252                 table[p][j] = i + 1;
253             }
254             else if (first_rhs[i]['^']) {
255                 for (k = 0; k < TSIZE; ++k) {
256                     if (follow[pro[i].str[0] - 'A'][k]) {
257                         table[p][k] = i + 1;
258                     }
259                 }
260             }
261         }
262     }
263     k = 0;
264     for (i = 0; i < no_pro; ++i) {
265         if (i == 0 || (pro[i - 1].str[0] != pro[i].str[0])) {
266             printf("%-10c", pro[i].str[0]);
267             for (j = 0; j < TSIZE; ++j) {
268                 if (table[k][j]) {
269                     printf("%-10s", pro[table[k][j] - 1].str);
270                 }
271             }
272             else if (terminal[j]) {
273                 printf("%-10s", "");
274             }
275             ++k;
276         }
277     }
278 }
279 }
```

INPUT:

The screenshot shows a web-based IDE interface for an online GDB compiler. The URL is onlinegdb.com/edit/iJEvx24tr. The toolbar includes back, forward, and refresh buttons, along with a lightning bolt icon, a file icon, an upload icon, a green "Run" button, a blue "Debug" button, and a red "Stop" button. The code editor has tabs for "main.c" and "input.txt", with "input.txt" currently selected. The input file contains the following grammar rules:

```
1 E->TA
2 A->+TA|^
3 T->FB
4 B->*FB|^
5 F->t|(E)
```

OUTPUT:

The screenshot shows the output of the program in the same online GDB compiler. The output consists of several parts:

- Initial grammar rules:

```
E->TA
A->+TA|^
T->FB
B->*FB|^
F->t|(E)
```
- FIRST sets:

```
FIRST OF E: ( t
FIRST OF A: + ^
FIRST OF T: ( t
FIRST OF B: * ^
FIRST OF F: ( t
```
- FOLLOW sets:

```
FOLLOW OF E: $ )
FOLLOW OF A: $ )
FOLLOW OF T: $ ) +
FOLLOW OF B: $ ) +
FOLLOW OF F: $ ) * +
```
- FIRST sets for non-terminals:

```
FIRST OF E->TA: ( t
FIRST OF A->+TA: +
FIRST OF A->^: ^
FIRST OF T->FB: ( t
FIRST OF B->*FB: *
FIRST OF B->^: ^
FIRST OF F->t: t
FIRST OF F->(E): (
```
- Parsing table header:

```
***** LL(1) PARSING TABLE *****
```
- Parsing table body:

	\$	()	*	+	t
E		E->TA				E->TA
A	A->^		A->^		A->+TA	
T		T->FB				T->FB
B	B->^		B->^	B->*FB	B->^	

Result : The program for Construction of Predictive Parsing Table (LL(1)) was executed successfully 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.

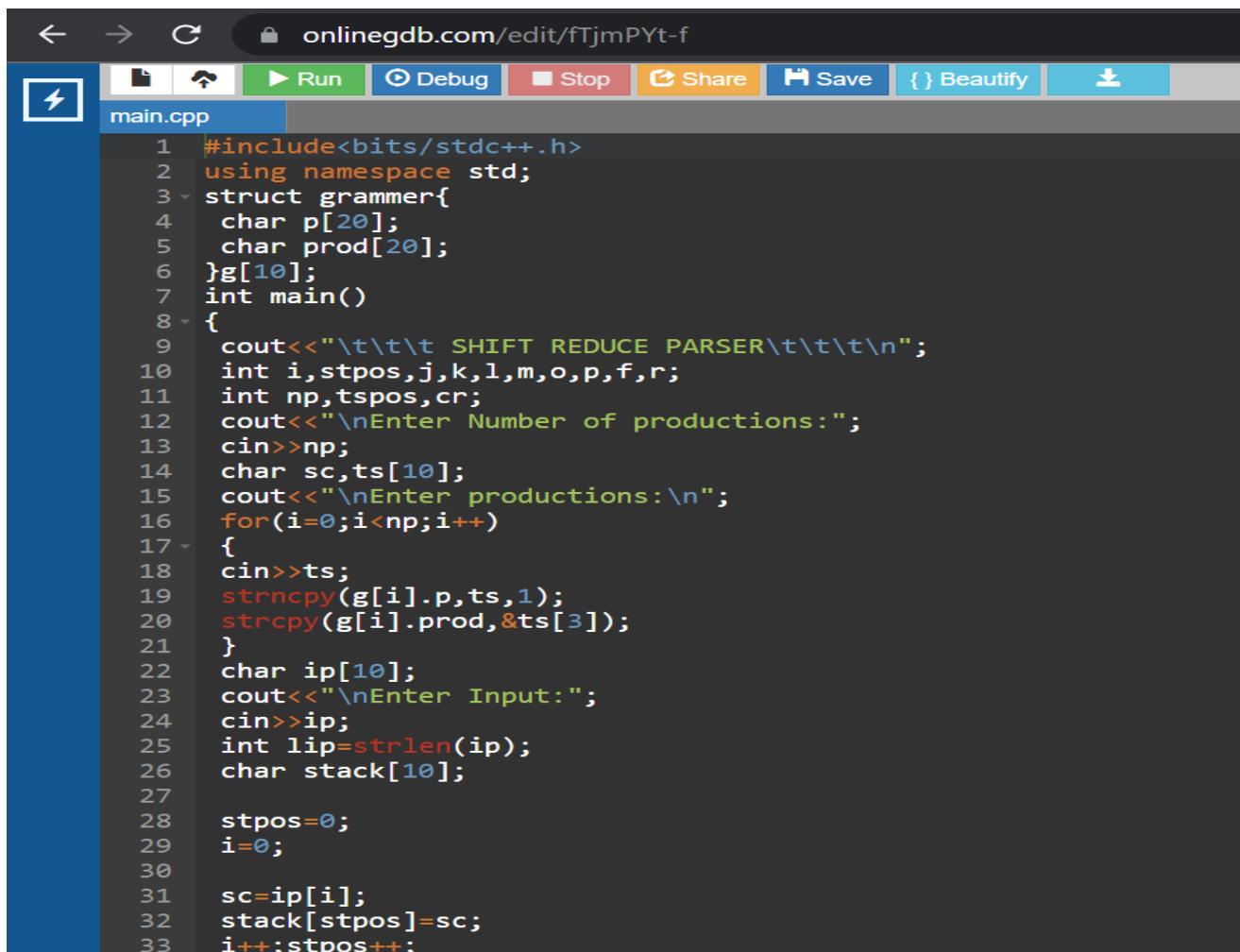
Calculator :- Let's consider the grammar.

$$\begin{aligned} E &\rightarrow 2E2 \\ E &\rightarrow 3E3 \\ E &\rightarrow 4 \end{aligned}$$

$$\begin{aligned} \text{INPUT} &\rightarrow 32423 \\ \text{String} & \end{aligned}$$

Stack	Input buffer	Parsing Action
\$	3 2 4 2 3 \$	Shift
\$ 3	2 4 2 3 \$	Shift
\$ 3 2	4 2 3 \$	Shift
\$ 3 2 4	2 3 \$	Reduce by E → 4
\$ 3 2 E	2 3 \$	shift
\$ 3 2 E 2	3 \$	Reduce by E → 2 E 2
\$ 3 E	3 \$	Shift
\$ 3 E 3	\$	Reduce by E → 3
\$ E	\$	Accept.

PROGRAM:



```

1 #include<bits/stdc++.h>
2 using namespace std;
3 struct grammar{
4     char p[20];
5     char prod[20];
6 }g[10];
7 int main()
8 {
9     cout<<"\t\t\t SHIFT REDUCE PARSER\t\t\t\n";
10    int i,stpos,j,k,l,m,o,p,f,r;
11    int np,tspos,cr;
12    cout<<"\nEnter Number of productions:";
13    cin>>np;
14    char sc,ts[10];
15    cout<<"\nEnter productions:\n";
16    for(i=0;i<np;i++)
17    {
18        cin>>ts;
19        strncpy(g[i].p,ts,1);
20        strcpy(g[i].prod,&ts[3]);
21    }
22    char ip[10];
23    cout<<"\nEnter Input:";
24    cin>>ip;
25    int lip=strlen(ip);
26    char stack[10];
27
28    stpos=0;
29    i=0;
30
31    sc=ip[i];
32    stack[stpos]=sc;
33    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';
```

```

68    }
69    tspos=0;
70    for(l=k;l<stpos;l++)
71    {
72        ts[tspos]=stack[l];
73        tspos++;
74    }
75
76    for(m=0;m<np;m++)
77    {
78        cr = strcmp(ts,g[m].prod);
79
80        if(cr==0)
81        {
82            for(l=k;l<10;l++)
83            {
84                stack[l]='\0';
85                stpos--;
86            }
87            stpos=k;
88
89            strcat(stack,g[m].p);
90            stpos++;
91            r=2;
92        }
93    }
94}
95}

96
97 sc=ip[i];
98 stack[stpos]=sc;
99 i++;stpos++;
100 }while(strlen(stack)!=1 && stpos!=lip);
101 if(strlen(stack)==1)

102 {
103     cout<<"\n\n \t\t\tSTRING IS ACCEPTED\t\t\t";
104 }
105 else
106     cout<<"\n\n \t\t\tSTRING IS REJECTED\t\t\t";
107     return 0;
108 }
109

```

INPUT:

SHIFT REDUCE PARSER

Enter Number of productions:4

Enter productions:

 $E \rightarrow E+E$
 $E \rightarrow E * E$
 $E \rightarrow (E)$
 $E \rightarrow a$

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 successfully in online GDB compiler.

WEEK-8 Computation of LEADING AND TRAILING

Exp - 8

28/03/22

Pranjay Poddar
RA1911028010129
J1 CSE-CCComputation 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) i.e. $A \rightarrow yas$ where y is E 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) i.e. $(A) \rightarrow yaS$ where S is E or any non-terminal
- (2) If 'a' is in terminal trailing (B) & $A \rightarrow B$, then 'a' is in trailing (A).

No. of Productions = 2

Variable = S, T

No of terminals = 5

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

S's production -

$$S \rightarrow S + T \mid (T)$$

$$T \rightarrow *, ,$$

$$\text{Lead}(S) = \{ (, +, *, , \}$$

$$\text{Lead}(T) = \{ *, , \}$$

$$\text{Trail}(S) = \{), +, *, , \}$$

$$\text{Trail}(T) = \{ *, , \}$$

Original
Version
of
Assignment
28/3/22

PROGRAM:

```
main.cpp
1 #include<iostream>
2 #include<conio.h>
3 #include<stdio.h>
4 #include<string.h>
5 #include<stdlib.h>
6 using namespace std;
7
8 int vars,terms,i,j,k,m,rep,count,temp=-1;
9 char var[10],term[10],lead[10][10],trail[10][10];
10 struct grammar
11 {
12     int prodno;
13     char lhs,rhs[20][20];
14 }gram[50];
15 void get()
16 {
17     cout<<"\nLEADING AND TRAILING\n";
18     cout<<"\nEnter the no. of variables : ";
19     cin>>vars;
20     cout<<"\nEnter the variables : \n";
21     for(i=0;i<vars;i++)
22     {
23         cin>>gram[i].lhs;
24         var[i]=gram[i].lhs;
25     }
26     cout<<"\nEnter the no. of terminals : ";
27     cin>>terms;
28     cout<<"\nEnter the terminals : ";
29     for(j=0;j<terms;j++)
30         cin>>term[j];
31     cout<<"\nPRODUCTION DETAILS\n";
32     for(i=0;i<vars;i++)
33     {
34         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                 {

```

```

69             if(gram[i].rhs[j][0]==var[m])
70             {
71                 temp=m;
72                 goto out;
73             }
74         out:
75         for(k=0;k<terms;k++)
76         {
77             if(lead[temp][k]==1)
78                 lead[i][k]=1;
79         }
80     }
81 }
82 }
83 }
84 }
85 void trailing()
86 {
87     for(i=0;i<vars;i++)
88     {
89         for(j=0;j<gram[i].prodno;j++)
90         {
91             count=0;
92             while(gram[i].rhs[j][count]!='\x0')
93                 count++;
94             for(k=0;k<terms;k++)
95             {
96                 if(gram[i].rhs[j][count-1]==term[k])
97                     trail[i][k]=1;
98                 else
99                 {
100                     if(gram[i].rhs[j][count-2]==term[k])
101                         trail[i][k]=1;
102                 }

```

```

103         }
104     }
105 }
106 for(rep=0;rep<vars;rep++)
107 {
108     for(i=0;i<vars;i++)
109     {
110         for(j=0;j<gram[i].prodno;j++)
111         {
112             count=0;
113             while(gram[i].rhs[j][count]!='\x0')
114                 count++;
115             for(m=1;m<vars;m++)
116             {
117                 if(gram[i].rhs[j][count-1]==var[m])
118                     temp=m;
119             }
120             for(k=0;k<terms;k++)
121             {
122                 if(trail[temp][k]==1)
123                     trail[i][k]=1;
124             }
125         }
126     }
127 }
128 }
129 void display()
130 {
131     for(i=0;i<vars;i++)
132     {
133         cout<<"\nLEADING("<<gram[i].lhs<<") = ";
134         for(j=0;j<terms;j++)
135         {
136             if(lead[i][j]==1)

```

```
137         cout<<term[j]<<",";
138     }
139 }
140 cout<<endl;
141 for(i=0;i<vars;i++)
142 {
143     cout<<"\nTRAILING("<<gram[i].lhs<<") = ";
144     for(j=0;j<terms;j++)
145     {
146         if(trail[i][j]==1)
147             cout<<term[j]<<",";
148     }
149 }
150 }
151 int main()
152 {
153
154     get();
155     leading();
156     trailing();
157     display();
158 }
159 }
```

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*T
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 successfully in online GDB compiler.

WEEK-8 Computation of LR(0) itemsExp - 9

28/03/22

Pranjay Poddar
 RA1911028010129
 CSE-CC J1

Computing of LR(0)

Aim:- To produce augmented grammar and get closure to obtain LR(0).

Algorithm:-

- ① Create augmented grammar.
 - 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 go to table where action specific the non-terminals and go to 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 .

$$S \rightarrow A A$$

$$A \rightarrow a A / b$$

$$S' \rightarrow \cdot S$$

$$S \rightarrow \cdot AA$$

$$A \rightarrow \cdot aA$$

$$A \rightarrow \cdot b$$

$$I_0 = \text{closure} (S' \rightarrow \cdot S)$$

$$= S' \rightarrow \cdot S$$

$$S \rightarrow \cdot AA$$

$$A \rightarrow \cdot aA$$

$$A \rightarrow \cdot b$$

$$I_1 = (I_0, S) = \text{closure}(S' \rightarrow S \cdot) = S' \rightarrow S \cdot$$

$$I_2 = (S_0, A) = \text{closure}(S \rightarrow A \cdot A)$$

$$I_3 = (I_2, a) = \text{closure}(A \rightarrow a \cdot A)$$

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

$$I_4 = (I_0, b) = \text{closure}(A \rightarrow b) = A \rightarrow b$$

$$I_5 = (I_2, A) = \text{closure}(S \rightarrow A \cdot A) = S \rightarrow A_2$$

$$I_6 = (I_1, A) = \text{closure}(A \rightarrow aA) = A \rightarrow aA$$

I_6 contains 7 states.

LR(0) Table

States	Actions			Notes.
	a	b	\$	A S
I_0	s_3	s_4	accept	$A_2 \quad 1$
I_1				
I_2				
I_3	s_3	s_4		5
I_4	s_3	s_4		6
I_5	γ_3	γ_3	γ_3	already
I_6	γ_1	γ_1	γ_1	$AA \leftarrow 2$
	γ_2	γ_2	γ_2	$1 A_0 \leftarrow A$

PROGRAM:

```
main.cpp
1 #include<iostream>
2 #include<conio.h>
3 #include<string.h>
4
5 using namespace std;
6
7 char prod[20][20],listofvar[26]={"ABCDEFGHIJKLMNPQR"};
8 int novar=1,i=0,j=0,k=0,n=0,m=0,arr[30];
9 int noitem=0;
10
11 struct Grammar
12 {
13     char lhs;
14     char rhs[8];
15 }g[20],item[20],clos[20][10];
16
17 int isvariable(char variable)
18 {
19     for(int i=0;i<novar;i++)
20         if(g[i].lhs==variable)
21             return i+1;
22     return 0;
23 }
24 void findclosure(int z, char a)
25 {
26     int n=0,i=0,j=0,k=0,l=0;
27     for(i=0;i<arr[z];i++)
28     {
29         for(j=0;j<strlen(clos[z][i].rhs);j++)
30         {
31             if(clos[z][i].rhs[j]=='.'
32                 && clos[z][i].rhs[j+1]==a)
```

```

52
53     clos[noitem][n].lhs=clos[z][i].lhs;
54     strcpy(clos[noitem][n].rhs,clos[z][i].rhs);
55     char temp=clos[noitem][n].rhs[j];
56     clos[noitem][n].rhs[j]=clos[noitem][n].rhs[j+1];
57     clos[noitem][n].rhs[j+1]=temp;
58     n=n+1;
59   }
60 }
61 for(i=0;i<n;i++)
62 {
63   for(j=0;j<strlen(clos[noitem][i].rhs);j++)
64   {
65     if(clos[noitem][i].rhs[j]=='.'
66       && isvariable(clos[noitem][i].rhs[j+1])>0)
67     {
68       for(k=0;k<novar;k++)
69     {
70       if(clos[noitem][i].rhs[j+1]==clos[0][k].lhs)
71     {
72       for(l=0;l<n;l++)
73     {
74       if(clos[noitem][l].lhs==clos[0][k].lhs && strcmp(clos[noitem][l].rhs,clos[0][k].rhs)==0)
75         break;
76       if(l==n)
77     {
78       clos[noitem][n].lhs=clos[0][k].lhs;
79       strcpy(clos[noitem][n].rhs,clos[0][k].rhs);
80       n=n+1;
81     }
82     }
83   }
84 }
85   if(flag==1)
86   {
87     arr[noitem++]=n;
88   }
89 }
90
91 int main()
92 {
93   cout<<"ENTER THE PRODUCTIONS OF THE GRAMMAR(0 TO END) :\n";
94   do
95   {
96     cin>>prod[i++];
97   }while(strcmp(prod[i-1],"0")!=0);
98   for(n=0;n<i-1;n++)
99   {
100     m=0;

```

```

101     j=novar;
102     g[novar++].lhs=prod[n][0];
103     for(k=3;k<strlen(prod[n]);k++)
104     {
105         if(prod[n][k] != '|')
106             g[j].rhs[m++]=prod[n][k];
107         if(prod[n][k]=='|')
108         {
109             g[j].rhs[m]='\0';
110             m=0;
111             j=novar;
112             g[novar++].lhs=prod[n][0];
113         }
114     }
115 }
116 for(i=0;i<26;i++)
117     if(!isvariable(listofvar[i]))
118         break;
119     g[0].lhs=listofvar[i];
120     char temp[2]={g[1].lhs,'\'\0'};
121     strcat(g[0].rhs,temp);
122     cout<<"\n\n augmented grammar \n";
123     for(i=0;i<novar;i++)
124         cout<<endl<<g[i].lhs<<"->"<<g[i].rhs<<" ";
125
126     for(i=0;i<novar;i++)
127     {
128         clos[noitem][i].lhs=g[i].lhs;
129         strcpy(clos[noitem][i].rhs,g[i].rhs);
130         if(strcmp(clos[noitem][i].rhs,"ε")==0)
131             strcpy(clos[noitem][i].rhs,".");
132         else
133         {
134             for(int j=strlen(clos[noitem][i].rhs)+1;j>=0;j--)

```

```

135         clos[noitem][i].rhs[j]=clos[noitem][i].rhs[j-1];
136         clos[noitem][i].rhs[0]='.';
137     }
138 }
139 arr[noitem++]=novar;
140 for(int z=0;z<noitem;z++)
141 {
142     char list[10];
143     int l=0;
144     for(j=0;j<arr[z];j++)
145     {
146         for(k=0;k<strlen(clos[z][j].rhs)-1;k++)
147         {
148             if(clos[z][j].rhs[k]=='.')
149             {
150                 for(m=0;m<l;m++)
151                     if(list[m]==clos[z][j].rhs[k+1])
152                         break;
153                 if(m==l)
154                     list[l++]=clos[z][j].rhs[k+1];
155             }
156         }
157     }
158     for(int x=0;x<l;x++)
159         findclosure(z,list[x]);
160 }
161 cout<<"\n THE SET OF ITEMS ARE \n\n";
162 for(int z=0; z<noitem; z++)
163 {
164     cout<<"\n I"<<z<<"\n\n";
165     for(j=0;j<arr[z];j++)
166         cout<<clos[z][j].lhs<<"->"<<clos[z][j].rhs<<"\n";
167
168 }
```

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

E->E+.T

T->.T*F

T->.F

F->.(E)

F->.i

I7

T->T*.F

F->.(E)

F->.i

I8

F->(E.)

E->E.+T

I9

E->E+T.

T->T.*F

I10

T->T*F.

I11

F->(E) .

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

WEEK-10 Intermediate code generation – Postfix, Prefix

4/04/22

Pranjay Poddar

Ex - 10

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 precedence 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 ')' push 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 parenthesis.
- (11) Convert logic of infix to prefix 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 to the operator into the stack.
- (15) Repeat steps 6 to 9 until the stack is empty.

Manual Calculation:-

Infix to Prefix

Infix

C + D * E

Prefix

CDE * +

Infix to Postfix

Infix

C + D * E

Postfix

+ C * DE

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

With
Date
n/n/22

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         if (isalpha(infix[i]) || isdigit(infix[i]))  
32             output += infix[i];  
33         else if (isOperator(infix[i])) {  
34             if (char_stack.empty())  
35                 char_stack.push(infix[i]);  
36             else {  
37                 if (getPriority(char_stack.top()) < getPriority(infix[i]))  
38                     char_stack.push(infix[i]);  
39                 else if (getPriority(char_stack.top()) >= getPriority(infix[i]))  
40                     output += char_stack.top();  
41                     char_stack.pop();  
42             }  
43         }  
44     }  
45     while (!char_stack.empty())  
46         output += char_stack.top();  
47     return output;  
48 }
```

```
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    }
103
104    int main()
105    {
106        string s = ("A+B");
107        cout << "Prefix of expression is: " << infixToPrefix(s) << std::endl;
108        return 0;
109    }

```

OUTPUT:**Postfix:**

```
The postfix string is: AB+  
...Program finished with exit code 0  
Press ENTER to exit console.
```

Prefix:

```
Prefix of expression is: +AB  
...Program finished with exit code 0  
Press ENTER to exit console.
```

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

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

Experiment - 11

Date: - 4/4/11

Pranjay Poddar
RA1911028010129
CSE-CC J1Intermediate 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 $a ::= b \text{ op } c$ perform the various actions. These are as follows:-

- (1) **Invoke** :- a function gets reg to find the location L where the result of computation $b \text{ op } c$ should be stored.
- (2) Consult the address description for y to determine the value of y .
- (3) Generate the instructions $\text{op } z'$, 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 this can exit then the block.

Calculations :-

Let the expression bc : $a + b * c + d$ is

$$* * T_1 = B * C$$

$$T_2 = A + T_1$$

$$T_3 = T_2 + D **$$

$$\Rightarrow a = b + c - d$$

$$t_1 = b + c$$

$$t_2 = t_1 - d$$

$$a = t_2$$

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

Verifying
Q/Wn
2/Wn

PROGRAM:

main.c

```
1 #include<stdio.h>
2 #include<ctype.h>
3 #include<stdlib.h>
4 #include<string.h>
5 void small();
6 void dove(int i);
7 int p[5]={0,1,2,3,4},c=1,i,k,l,m,pi;
8 char sw[5]={ '=', '-' , '+' , '/' , '*' },j[20],a[5],b[5],ch[2];
9 void main()
10 {
11     printf("Enter the expression:");
12     scanf("%s",j);
13     printf("\tThe Intermediate code is:\n");
14     small();
15 }
16 void dove(int i)
17 {
18     a[0]=b[0]='\0';
19     if(!isdigit(j[i+2])&&!isdigit(j[i-2]))
20     {
21         a[0]=j[i-1];
22         b[0]=j[i+1];
23     }
24     if(isdigit(j[i+2])){
25         a[0]=j[i-1];
26         b[0]='t';
27         b[1]=j[i+2];
28     }
29     if(isdigit(j[i-2]))
30     {
31         b[0]=j[i+1];
32         a[0]='t';
33         a[1]=j[i-2];
34         b[1]='\0';
35 }
```

```

35 }
36 if(isdigit(j[i+2]) &&isdigit(j[i-2]))
37 {
38 a[0]='t';
39 b[0]='t';
40 a[1]=j[i-2];
41 b[1]=j[i+2];
42 sprintf(ch,"%d",c);
43 j[i+2]=j[i-2]=ch[0];
44 }
45 if(j[i]=='*')
46 printf("\tt%d=%s*s\n",c,a,b);
47 if(j[i]=='/')
48 printf("\tt%d=%s/%s\n",c,a,b);
49 if(j[i]=='+')
50 printf("\tt%d=%s+%s\n",c,a,b);if(j[i]=='-')
51 printf("\tt%d=%s-%s\n",c,a,b);
52 if(j[i]=='=')
53 printf("\tc=t%d",j[i-1],--c);
54 sprintf(ch,"%d",c);
55 j[i]=ch[0];
56 c++;
57 small();
58 }
59 void small()
60 {
61 pi=0;l=0;
62 for(i=0;i<strlen(j);i++)
63 {
64 for(m=0;m<5;m++)
65 if(j[i]==sw[m])
66 if(pi<=p[m])
67 {
68 pi=p[m];
69 l=1;
70 k=i;
71 }
72 }
73 if(l==1)
74 dove(k);
75 else
76 exit(0);}
```

OUTPUT:

```
Enter the expression:A+B*C-D
The Intermediate code is:
t1=B*C
t2=A+t1
t3=t1-D

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

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

WEEK-12 Implementation Of DAG

Exp 12

Date : 11/04/22

Pranjay Poddar

RA1911028010129

J₁ SectionImplementation of DAG

Aim:- A program to implement DAG.

Procedure :-

- (1) The leaves of a graph are loaded by unique identifier.
- (2) Interior nodes of the graph are loaded by an operator symbol.
- (3) Nodes are also given sequence of identifiers for tables to store the computed value.
- (4) If y operand is undefined then create node (y).
- (5) If z operand is undefined then for case (i) create node (z).
- (6) For case (ii), create node (op) whose right child is node (z) and left child [] is node (y).
- (7) For case (iii) check whether there is node (op) with one child node (y).

- (8) For case (iii) node n will be node (g).
- (9) For node (x) delete x from the list of identifiers.
- (10) Append α 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 : $x = y \text{ or } z$

Case 2 : $x = b + z$

Case 3 : $x = y$

Set the expression is $a = b + -c + b * -c$

$$z = b *$$

$$y = c + z$$

$$x = c - y$$

$$w = x - c$$

$$a = w \quad a = \$$$

Operational Diagram

Result : Successfully executed the program related to DAG.

PROGRAM:

main.cpp

```
1 #include<stdio.h>
2 #include<string.h>
3 int i=1,j=0,no=0,tmpch=90;
4 char str[100],left[15],right[15];
5 void findopr();
6 void explore();
7 void fleft(int);
8 void fright(int);
9 struct exp
10 {
11     int pos;
12     char op;
13 }k[15];
14 void main()
15 {
16     printf("\t\tINTERMEDIATE CODE GENERATION OF DAG\n\n");
17     scanf("%s",str);
18     printf("The intermediate code:\t\tExpression\n");
19     findopr();
20     explore();
21 }
22 void findopr()
23 {
24     for(i=0;str[i]!='\0';i++)
25         if(str[i]==':')
26     {
27         k[j].pos=i;
28         k[j++].op=':';
29     }
30     for(i=0;str[i]!='\0';i++)
31         if(str[i]=='/')
32     {
33         k[j].pos=i;
34         k[j++].op='/';
35     }
36 }
```

```

35 }
36 for(i=0;str[i]!='\0';i++)
37   if(str[i]=='*')
38   {
39     k[j].pos=i;
40     k[j++].op='*';
41   }
42 for(i=0;str[i]!='\0';i++)
43   if(str[i]=='+')
44   {
45     k[j].pos=i;
46     k[j++].op='+';
47   }
48 for(i=0;str[i]!='\0';i++)
49   if(str[i]=='-')
50   {
51     k[j].pos=i;
52     k[j++].op='-';
53   }
54 }
55 void explore()
56 {
57   i=1;
58   while(k[i].op!='\0')
59   {
60     fleft(k[i].pos);
61     fright(k[i].pos);
62     str[k[i].pos]=tmpch--;
63     printf("\t%c := %s%c%s\t\t",str[k[i].pos],left,k[i].op,right);
64     for(j=0;j <strlen(str);j++)
65       if(str[j]!='$')
66         printf("%c",str[j]);
67       printf("\n");
68   i++;

```

```

69 }
70 fright(-1);
71 if(no==0)
72 {
73     fleft(strlen(str));
74     printf("\t%s := %s",right,left);
75 }
76 printf("\t%s := %c",right,str[k[--i].pos]);
77 }
78 void fleft(int x)
79 {
80     int w=0,flag=0;
81     x--;
82     while(x!= -1 &&str[x]!='+' 
83     &&str[x]!='*'&&str[x]!='='&&str[x]!='\0'&&str[x]!='-'&&str[x]!='/'
84     &&str[x]!=':'){
85         if(str[x]!='$'&& flag==0)
86         {
87             left[w++]=str[x];
88             left[w]='\0';
89             str[x]='$';
90             flag=1;
91         }
92         x--;
93     }
94 }
95 void fright(int x)
96 {
97     int w=0,flag=0;
98     x++;
99     while(x!= -1 && str[x]!=
100     '+'&&str[x]!='*'&&str[x]!='\0'&&str[x]!='='&&str[x]!=':'&&str[x]!='-
101     '&&str[x]!='/')
102     {
103         if(str[x]!='$'&& flag==0)
104         {
105             right[w++]=str[x];
106             right[w]='\0';
107             str[x]='$';
108             flag=1;
109         }
110         x++;
111     }
112 }

```

OUTPUT:

```
INTERMEDIATE CODE GENERATION OF DAG  
input  
=b*-c+b*-c  
The intermediate code: Expression  
Z := b* a=b*-c+Z-c  
Y := c+Z a=b*-Y-c  
X := c-Y a=b*X-c  
W := X-c a=b*W  
a := W a := $  
  
.Program finished with exit code 0  
Press ENTER to exit console.
```

Result : The program Implementation of DAG was executed successfully 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 of the stack element will be displayed

Calculation :-

Enter your choice (1-5) : ,

Elements = 10

enter more elements : y

SP Verified
Done
11/4/22

PROGRAM:

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #define TRUE 1
4 #define FALSE 0
5 typedef struct Heap
6 {
7     int data;
8     struct Heap *next;
9 }
10 node;
11 node *create();
12 void main()
13 {
14     int choice, val;
15     char ans;
16     node *head;
17     void display(node *);
18     node *search(node *, int);
19     node *insert(node *);
20     void dele(node **);
21     head=NULL;
22     do
23     {
24         printf("\nprogram to perform various operations on heap using dynamic memory management");
25         printf("\n1.create");
26         printf("\n2.display");
27         printf("\n3.insert an element in a list");
28         printf("\n4.delete an element from list");
29         printf("\n5.quit");
30         printf("\nEnter your choice(1-5)");
31         scanf("%d", &choice);
32         switch(choice)
33         {
34             case 1: head=create();
35             break;
36             case 2: display(head);
37             break;
38             case 3: head=insert(head);
39             break;
40         }
41     } while(ans=='y' || ans=='Y');
42 }
```

```

40 break;
41 case 4:delete(&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 }

```

```

79 while(ans=='y');
80 printf("\nthe list is created\n");
81 return head;
82 }
83 node *get_node()
84 {
85 node *temp;
86 temp=(node*)malloc(sizeof(node));
87 temp->next=NULL;
88 return temp;
89 }
90 void display(node *head)
91 {
92 node *temp;
93 temp=head;
94 if(temp==NULL)
95 {
96 printf("\nthe list is empty\n");
97 return;
98 }
99 while(temp!=NULL)
100 {
101 printf("%d->",temp->data);
102 temp=temp->next;
103 }
104 printf("NULL");
105 }
106 node *search(node *head,int key)
107 {
108 node *temp;
109 int found;
110 temp=head;
111 if(temp==NULL)
112 {
113 printf("the linked list is empty\n");
114 return NULL;
115 }
116 found=FALSE;
117 while(temp!=NULL && found==FALSE)

```

```

118 {
119 if(temp->data!=key)
120 temp=temp->next;
121 else
122 found=TRUE;
123 }
124 if(found==TRUE)
125 {
126 printf("\nthe element is present in the list\n");
127 return temp;
128 }
129 else
130 {
131 printf("the element is not present in the list\n");
132 return NULL;
133 }
134 }
135 node *insert(node *head)
136 {
137 int choice;
138 node *insert_head(node *);
139 void insert_after(node *);
140 void insert_last(node *);
141 printf("n1.insert a node as a head node");
142 printf("n2.insert a node as a head node");
143 printf("n3.insert a node at intermediate position in t6he list");
144 printf("\nEnter your choice for insertion of node:");
145 scanf("%d",&choice);
146 switch(choice)
147 {
148 case 1:head=insert_head(head);
149 break;
150 case 2:insert_last(head);
151 break;
152 case 3:insert_after(head);
153 break;
154 }
155 return head;
156 }

```

```

157 node *insert_head(node *head)
158 {
159     node *New,*temp;
160     New=get_node();
161     printf("\nEnter the element which you want to insert");
162     scanf("%d",&New->data);
163     if(head==NULL)
164         head=New;
165     else
166     {
167         temp=head;
168         New->next=temp;
169         head=New;
170     }
171     return head;
172 }
173 void insert_last(node *head)
174 {
175     node *New,*temp;
176     New=get_node();
177     printf("\nEnter the element which you want to insert");
178     scanf("%d",&New->data);
179     if(head==NULL)
180         head=New;
181     else
182     {
183         temp=head;
184         while(temp->next!=NULL)
185             temp=temp->next;
186         temp->next=New;
187         New->next=NULL;
188     }
189 }
190 void insert_after(node *head)
191 {
192     int key;
193     node *New,*temp;
194     New=get_node();
195     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("\nEnter 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;

```

```

235 }
236 else
237 flag=TRUE;
238 }
239 if(flag)
240 return prev;
241 else
242 return NULL;
243 }
244 void dele(node **head)
245 {
246 node *temp,*prev;
247 int key;
248 temp=*head;
249 if(temp==NULL)
250 {
251 printf("\nthe list is empty\n");
252 return;
253 }
254 printf("\nEnter the element you want to delete:");
255 scanf("%d",&key);
256 temp=search(*head,key);
257 if(temp!=NULL)
258 {
259 prev=get_prev(*head,key);
260 if(prev!=NULL)
261 {
262 prev->next=temp->next;
263 free(temp);
264 }
265 else
266 {
267 *head=temp->next;
268 free(temp);
269 }
270 printf("\nthe element is deleted\n");
271 }
272 }

```

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 choice(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 the 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 choice(1-5)
```

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