

18CSC304J
Compiler Design
Portfolio
SEMESTER – VI



Name of the Student: KAUSHIK TAYI

Register Number: RA2011030010048

DEPARTMENT OF NETWORKING AND
COMMUNICATIONS

SCHOOL OF COMPUTING

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
(Deemed University u/s 3 of UGC Act 1956)
Kattankulathur, Chengalpattu District, 603 202.



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

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR - 603 203**

BONAFIDE CERTIFICATE

Certified that this is the bonafide work of KAUSHIK TAYI (RA2011030010048) who carried out the portfolio, mini project work and Laboratory exercises under my supervision for 18CSC304J – COMPILER DESIGN. Certified further, that to the best of my knowledge the work reported herein does not form part of any other work.

Dr. B Yamini
ASSISTANT PROFESSOR
18CSC204J -Design and Analysis of Algorithms
Course Faculty

Department of Networking and Communications

Dr. Annapurani.K
HEAD OF THE DEPARTMENT
Department of Networking and
Communications

Signature of the Internal Examiner-I

Signature of the Internal Examiner-II

INDEX



Name: Kaushik Tanyi Class: 3rd Subject: C.D.

S.No.	Date	Topic	Page No.	Signature
1	26/1/23	Exp-1: Implementation of lexical Analyzer	10	J
2	26/1/23	Exp-2: R.E to NFA conversion	10	J
3	5/2/23	Program to implement the conversion from NFA to DFA	9	J
4	8/2/23	Elimination of left recursion to left factoring	5/6	J
5	21/2/23	First and Follow	10/8	J
6	22/2/23	Predictive parser	8	J
7	9/3/23	Shift Reduce Parser	8	J
8	9/3/23	Leading & Trailing	8	J
9	16/3/23	Computation of LR(0) items	10/8	J
10	22/2/23	Suffix to prefix & postfix	7	J
11	31/3/23	Autosm - code generation	8/9	J
12	10/4/23	Simple Code Gen.	7/8	J
13	18/4/23	Implementation of DAB.	7	J

Completed
J
20/4/23

* Compiler Design. RA2011030610048

* Assignment 02

① Check whether the given grammar is
Trace $id=id$ if there is any conflict
Solve with multiple entry.

$$S \rightarrow L = R$$

$$S \rightarrow R$$

$$L \rightarrow * R$$

$$L \rightarrow id$$

$$R \rightarrow L$$

\Rightarrow A1: step 1: write the assignment
grammar.

$$\Rightarrow S' \rightarrow \#$$

$$S' \rightarrow L = R$$

$$S \rightarrow R$$

$$L \rightarrow * R$$

$$L \rightarrow id$$

$$R \rightarrow L$$

\Rightarrow step-2: find the set of them based
on closer operation and goto

$$\underline{I_0}: S' \rightarrow \cdot S \quad L \rightarrow \cdot id$$

$$S \rightarrow \cdot L = R \quad R \rightarrow \cdot L$$

$$S \rightarrow \cdot R$$

$$L \rightarrow \cdot * R$$

I₀

$$\text{Goto}(0, S) = I_1$$

$$\text{Goto}(0, S) = 1$$

$$\text{Goto}(0, L) = I_2$$

$$\text{Goto}(0, L) = 2$$

$$\text{Goto}(0, R) = I_3$$

$$\text{Goto}(0, R) = 3$$

$$\text{Goto}(0, \#) = I_4$$

$$\text{Action}(0, \#) = S_4$$

$$\text{Goto}(0, id) = I_5$$

$$\text{Action}(0, id) = S_5$$

I₁:

$$S' \rightarrow S \quad \text{Action}(1, \$) = \text{accept.}$$

I₂:

$$R \rightarrow L$$

$$\text{Action}(2, \$) = r_5$$

$$\text{Action}(2, \$) = r_5$$

I₃:

$$S \rightarrow R$$

$$\text{Action}(3, \$) = r_2$$

I₄:

$$\text{Goto}(4, R) = I_7$$

$$\text{Goto}(4, R) = 7$$

$$\text{Goto}(4, L) = I_8$$

$$\text{Goto}(4, L) = 8$$

$$\text{Goto}(4, \#) = I_4$$

$$\text{Action}(4, \#) = S_4$$

$$\text{Goto}(4, id) = I_5$$

$$\text{Action}(4, id) = S_5$$

I₅:

$$L \rightarrow id$$

I₆:

$$\text{Goto}(6, R) = I_9$$

$$\text{Goto}(6, R) = 9$$

$$\text{Goto}(6, L) = I_8$$

$$\text{Goto}(6, L) = 8$$

$$\text{Goto}(6, \#) = I_4$$

$$\text{Action}(6, \#) = S_4$$

$$\text{Goto}(6, id) = I_5$$

$$\text{Action}(6, id) = S_5$$

$\Rightarrow \underline{I_9}$

$L \rightarrow *R$

Action($*$, $=$) = r_3

Action($*$, $\$$) = r_3

$\Rightarrow \underline{I_8}$

$R \rightarrow L^*$

action($*$, $=$) = r_3

action($*$, $\$$) = r_3

$\Rightarrow \underline{I_9}$

$S \rightarrow L = R$

action($*$, $\$$) = r_1

State	Action				goto		
	=	*	id	\$	S	L	R
0		S_u	$S_{\bar{u}}$		1	2	3
1				accept			
2	$S_u R_{\bar{u}}$			r_5			
3							
4		S_u	$S_{\bar{u}}$			8	9
5	r_u			r_u			
6		S_u	$S_{\bar{u}}$			8	9
7	r_3			r_3			
8	r_5			r_5			
9				r_1			

$id = id$

<u>Stack</u>	<u>Input</u>	<u>Comment</u>
0	$id = id \$$	S_5
0 id S	$= id \$$	$r_u(L \rightarrow id)$
0 L 2	$= id \$$	$r_s(S \rightarrow L)$
0 L 3	$= id \$$	Rejected

considering
reduce

<u>Stack</u>	<u>Input</u>	<u>Comment</u>
0	$id = id \$$	S_5
0 id S	$= id \$$	$r_u(L \rightarrow id)$
0 L 2	$= id \$$	S_6
0 L 2 = b	$id \$$	S_5
0 L 2 = b id S	$\$$	$r_u(L \rightarrow id)$
0 L 2 = b L 8	$\$$	$r_s(R \rightarrow L)$
0 L 2 = b L 9	$\$$	$r_{ils} \rightarrow L = R)$
0 S 1	$\$$	Accepted.

considering increase

— X —

Assignment . 01

Q) To check whether the given grammar is LL(1)

$$A) \quad S \rightarrow iEtS / iEtSS / a$$

$$E \rightarrow b$$

⇒ After LR

$$S \rightarrow iEtSS' / a$$

$$S' \rightarrow eS / e$$

$$E \rightarrow b$$

⇒ Step-3 : find First and Follow of each NT in grammar obtained.

$$\Rightarrow \text{FIRST}(S) = \{i, a\}$$

$$\text{Follow} = \{e, b\}$$

$$\text{FIRST}(S') = \{e, e\}$$

$$\text{Follow} = \{e, b\}$$

$$\text{FIRST}(E) = \{b\}$$

$$\text{Follow} = \{a\}$$

⇒ Step-4 . Construct the parsing table.

$$S \rightarrow iEtSS'$$

$$M[S, i] = S \rightarrow iEtSS'$$

$$S' \rightarrow eS$$

$$M[S', e] = S' \rightarrow eS$$

$$E \rightarrow b$$

$$M[E, b] = E \rightarrow b$$

$$S \rightarrow a$$

$$M[S, a] = S \rightarrow a$$

$$S' \rightarrow e$$

$$\text{Follow}(S') = \{e, b\}$$

$$\therefore M[S', e]$$

$$S' \rightarrow e$$

$$M[S', b]$$

$$S \rightarrow e$$

	a	b	c	i	+	⋄
S	$S \rightarrow a$			$S \rightarrow i S$		
S'			$S' \rightarrow c S$			$S' \rightarrow \epsilon$
E		$E \rightarrow b$				

- ② To check whether the given grammar is or not of LR(0), Trace input: (a, c)

$$S \rightarrow (L) / a$$

$$L \rightarrow L, S / S$$

- A) ① Elimination of LR

$$L \rightarrow L, S / S$$

$$L \rightarrow S L'$$

$$L' \rightarrow , S L' / \epsilon$$

$$S \rightarrow (L) / a$$

- ② Left factoring is not needed

- ③ First and Follow

$$\text{FIRST}(L) = \{ (, a \}$$

$$\text{FIRST}(L') = \{ ,, \epsilon \}$$

$$\text{FIRST}(S) = \{ (, a \}$$

$$\text{FOLLOW}(L) = \{ \}$$

$$\text{FOLLOW}(L') = \{ \}$$

$$\text{FOLLOW}(S) = \{ \}$$

- ④ To construct Parsing table m

	a	()	,	\$
S	$S \rightarrow a$	$S \rightarrow (L)$			
L	$L \rightarrow S L'$	$L \rightarrow S L'$			
L'			$L' \rightarrow \epsilon$	$L' \rightarrow ,$	

stack

input buffer

Parsing action

\$
\$ L
\$ (a
\$ (s
\$ (L
\$ (L,
\$ (L,a
\$ (L,a s
\$ (L
\$ (L)
\$ S

(a,a)\$
\$ a,a)\$
),a)\$
),a)\$
),a)\$
a)\$
)\$
)\$
)\$
\$
\$

Shift
Shift
Reduce $s \rightarrow a$
Reduce $L \rightarrow s$
Shift
Shift
Reduce $s \rightarrow a$
Reduce $s \rightarrow (L, s$
Shift
Reduce $s \rightarrow (L)$
Accept



SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that this mini project report titled c+ is the bonafide work done by ADURTHI ADITYA ANSHU (RA2011030010054) and KAUSHIK TAYI (RA2011030010048) who carried out the mini project work and Laboratory exercises under my supervision for **18CSC304J – COMPILER DESIGN**. Certified further, that to the best of my knowledge, the work reported herein does not form part of any other work.

Dr. B Yamini
15/5/23
Dr. B Yamini
ASSISTANT PROFESSOR
18CSC204J -Design and Analysis of Algorithms
Course Faculty
Department of Networking and Communications



Dr. Annapurani.K
Dr. Annapurani.K
HEAD OF THE DEPARTMENT
Department of Networking and
Communications

Signature of the Internal Examiner-I

Signature of the Internal Examiner-I

Experiment-1

Implementation of Lexical Analyzer in CPP

Aim:

To write a program implementing the Lexical Analyzer using C++.

Procedure:

1. Take the input from the .txt file.
2. Check for keywords present in the input and print it.
3. Using for loop check for the operators present in the input and print it.
4. Using for loop check for all the symbols present in the input and print it.
5. Using for loop check for all the symbols present in the input and print it.
6. Also, check for all the constants and identifiers present in the input and print it.

Code:

```
#include <bits/stdc++.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() {
    system("cls");
    int tk = 0;
    char ch, buffer[15], operators[] = "+-*/%=";
    ifstream fin("W1.txt");
    int i, j = 0;
    if (!fin.is_open()){
        cout << "error while opening the file\n";
        exit(0);
    }
    while (!fin.eof()) {
        ch = fin.get();
        for (i = 0; i < 6; ++i) {
            if (ch == operators[i]) {
                cout << ch << " is operator\n";
                tk++;
            }
        }
    }
}

```

```

    }
    if (isalnum(ch) {
        buffer[j++] = ch;
    }
    else if ((ch == ' ' || ch == '\n') && (j != 0)) {
        buffer[j] = '\0';
        j = 0;
        if (isKeyword(buffer) == 10) {
            cout << buffer << " is keyword\n";
            tk++;
        }
        else {
            cout << buffer << " is identifier\n";
            tk++;
        }
    }
}

}

fin.close();

cout << "\nTotal number of tokens present in the 'W1.txt' file is: " << tk <<
"\n\n";

system("pause");

return 0;

}

```

Text File:

```
int main() {  
    int r;  
    float pi = 3.14;  
    cin >> r;  
    float area;  
    area = pi * r * r;  
    cout << area;  
    return 0;  
}
```

Output:

```
int is keyword  
main is identifier  
int is keyword  
r is identifier  
float is keyword  
pi is identifier  
= is operator  
314 is identifier  
cin is identifier  
r is identifier  
float is keyword  
area is identifier  
area is identifier  
= is operator  
pi is identifier  
* is operator  
r is identifier  
* is operator  
r is identifier  
cout is identifier  
area is identifier  
return is keyword  
0 is identifier  
  
Total number of tokens present in the 'w1.txt' file is: 23  
  
Press any key to continue . . . █
```

Result:

Implementation of Lexical Analyzer using C++ has been done successfully.

Experiment-2 Conversion from Regular Expression to NFA

Aim:

To write a program for converting Regular Expression to NFA.

Procedure:

1. Start
2. Get the input from the user
3. Initialize separate variables and functions for Postfix, Display and NFA
4. Create separate methods for different operators like +, *,.
5. By using Switch case Initialize different cases for the input
6. For '.' Operator Initialize a separate method by using various stack functions do the same for the other operators like '*' and '+ '.
7. Regular expression is in the form like a.b (or) a+b
8. Display the output
9. Stop

Code:

```
#include <bits/stdc++.h>

int main()
{
    system("cls");
    char reg[20];
```



```

int q[20][3], i, j, len, a, b;
for (a = 0; a < 20; a++)
{
    for (b = 0; b < 3; b++)
    {
        q[a][b] = 0;
    }
}

printf("Regular expression: \n");
scanf("%s", reg);
len = strlen(reg);
i = 0;
j = 1;
while (i < len)
{
    if (reg[i] == 'a' && reg[i + 1] != '/' && reg[i + 1] != '*')
    {
        q[j][0] = j + 1;
        j++;
    }
    if (reg[i] == 'b' && reg[i + 1] != '/' && reg[i + 1] != '*')
    {
        q[j][1] = j + 1;
        j++;
    }
    if (reg[i] == 'e' && reg[i + 1] != '/' && reg[i + 1] != '*')
    {
        q[j][2] = j + 1;
        j++;
    }
}

```

```
if (reg[i] == 'a' && reg[i + 1] == '/' && reg[i + 2] == 'b')
```

```
{
```

```
    q[j][2] = ((j + 1) * 10) + (j + 3);
```

```
    j++;
```

```
    q[j][0] = j + 1;
```

```
    j++;
```

```
    q[j][2] = j + 3;
```

```
    j++;
```

```
    q[j][1] = j + 1;
```

```
    j++;
```

```
    q[j][2] = j + 1;
```

```
    j++;
```

```
    i = i + 2;
```

```
}
```

```
if (reg[i] == 'b' && reg[i + 1] == '/' && reg[i + 2] == 'a')
```

```
{
```

```
    q[j][2] = ((j + 1) * 10) + (j + 3);
```

```
    j++;
```

```
    q[j][1] = j + 1;
```

```
    j++;
```

```
    q[j][2] = j + 3;
```

```
    j++;
```

```
    q[j][0] = j + 1;
```

```
    j++;
```

```
    q[j][2] = j + 1;
```

```
    j++;
```

```
    i = i + 2;
```

```
}
```

```
if (reg[i] == 'a' && reg[i + 1] == '*')
```

```
{
```

```

    q[j][2] = ((j + 1) * 10) + (j + 3);
    j++;
    q[j][0] = j + 1;
    j++;
    q[j][2] = ((j + 1) * 10) + (j - 1);
    j++;
}
if (reg[i] == 'b' && reg[i + 1] == '*')
{
    q[j][2] = ((j + 1) * 10) + (j + 3);
    j++;
    q[j][1] = j + 1;
    j++;
    q[j][2] = ((j + 1) * 10) + (j - 1);
    j++;
}
if (reg[i] == ')' && reg[i + 1] == '*')
{
    q[0][2] = ((j + 1) * 10) + 1;
    q[j][2] = ((j + 1) * 10) + 1;
    j++;
}
i++;
}
printf("\nTransition function\n");
for (i = 0; i <= j; i++)
{
    if (q[i][0] != 0)
        printf("\n q[%d,a]-->%d", i, q[i][0]);
    if (q[i][1] != 0)

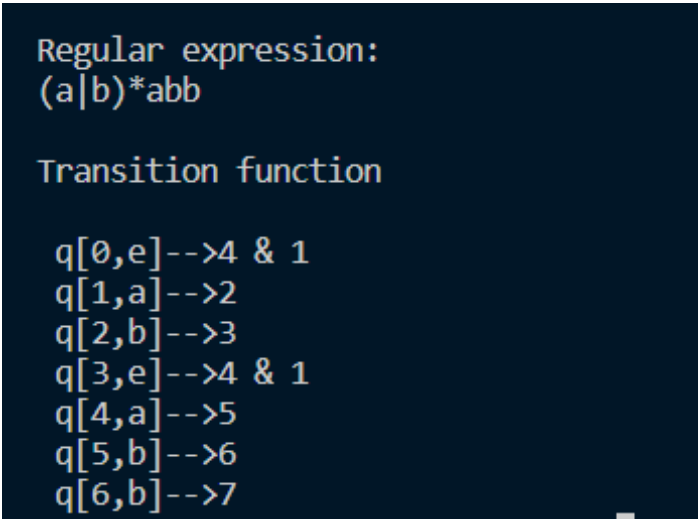
```

```

        printf("\n q[%d,b]-->%d", i, q[i][1]);
    if (q[i][2] != 0)
    {
        if (q[i][2] < 10)
            printf("\n q[%d,e]-->%d", i, q[i][2]);
        else
            printf("\n q[%d,e]-->%d & %d", i, q[i][2] / 10, q[i][2] % 10);
    }
}
printf("\n");
system("pause");
return 0;
}

```

Output:



```

Regular expression:
(a|b)*abb

Transition function

q[0,e]-->4 & 1
q[1,a]-->2
q[2,b]-->3
q[3,e]-->4 & 1
q[4,a]-->5
q[5,b]-->6
q[6,b]-->7

```

Result:

Implementation of a program for converting Regular Expression to NFA.

has been done successfully.

Experiment-3

CONVERSION OF NFA TO DFA

Aim:

To write a program for converting NFA to DFA.

Procedure:

1. Start
2. Get the input from the user
3. Set the only state in SDFA to “unmarked”.
4. While SDFA contains an unmarked state do:
 - a. Let T be that unmarked state
 - b. for each a in % do $S = e\text{-Closure}(\text{MoveNFA}(T,a))$
 - c. if S is not in SDFA already then, add S to SDFA (as an “unmarked”)
 - d. Set $\text{MoveDFA}(T,a)$ to S.
5. For each S in SDFA if any s & S is a final state in the NFA then, mark S a final state in the DFA.
6. Print the result.

Code:

```
#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:

```
Enter the Regular Expression and Q to exit: a/b  
NFA : 0 e 1 a 2 e 5 0 e 3 b 4 e 5  
DFA : 1 a 2 3 b 4  
Enter the Regular Expression and Q to exit: Q  
NFA : 0 Q 1  
DFA : 0 Q 1
```

Result:

The given NFA was converted to a DFA successfully.

Experiment-4

ELIMINATION OF AMBIGUITY

Aim:

To write a program implementing elimination of ambiguity using Left Recursion and Left Factoring.

Procedure:

a. Elimination of Left Recursion:

1. Start the program.
2. Initialize the arrays for taking input from the user.
3. Prompt the user to input the no. of non-terminals having left recursion and no. of productions for these non-terminals.
4. Prompt the user to input the production for non-terminals.
5. Eliminate left recursion using the following rules:-

$$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \dots \mid A\alpha_m$$
$$A \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$$

Then replace it by

$$A \rightarrow \beta_i A' \quad i=1,2,3,\dots,m$$
$$A' \rightarrow \alpha_j \quad j=1,2,3,\dots,n$$
$$A' \rightarrow \epsilon$$

6. After eliminating the left recursion by applying these rules, display the productions without left recursion.

b. Implementation of Left Factoring:

1. Start
2. Ask the user to enter the set of productions
3. Check for common symbols in the given set of productions by comparing with:

$A \rightarrow aB1 | aB2$

4. If found, replace the particular productions with:

$A \rightarrow aA'$

$A' \rightarrow B1 | B2 | \epsilon$

5. Display the output
6. Exit

Code:

a. Elimination of Left Recursion:

```
#include <bits/stdc++.h>

using namespace std;

int main()
{
    system("cls");

    int n;

    cout << "\nEnter number of non terminals: ";

    cin >> n;

    cout << "\nEnter non terminals one by one: ";

    int i;

    vector<string> nonter(n);

    vector<int> leftrecr(n, 0);

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

    {

        cout << "\Non terminal " << i + 1 << " : ";
```



```

        cin >> nonter[i];
    }
    vector<vector<string>>> prod;
    cout << "\nEnter 'esp' for null";
    for (i = 0; i < n; ++i)
    {
        cout << "\nNumber of " << nonter[i] << " productions: ";
        int k;
        cin >> k;
        int j;
        cout << "\nOne by one enter all " << nonter[i] << " productions";
        vector<string> temp(k);
        for (j = 0; j < k; ++j)
        {
            cout << "\nRHS of production " << j + 1 << ": ";
            string abc;
            cin >> abc;
            temp[j] = abc;
            if (nonter[i].length() <= abc.length() && nonter[i].compare(abc.substr(0,
nonter[i].length())) == 0)
                leftrecr[i] = 1;
        }
        prod.push_back(temp);
    }
    for (i = 0; i < n; ++i)
    {
        cout << leftrecr[i];
    }
    for (i = 0; i < n; ++i)
    {
        if (leftrecr[i] == 0)

```

```

        continue;

    int j;

    nonter.push_back(nonter[i] + "");

    vector<string> temp;

    for (j = 0; j < prod[i].size(); ++j)

    {

        if (nonter[i].length() <= prod[i][j].length() && nonter[i].compare(prod[i][j].substr(0,
nonter[i].length())) == 0)

        {

            string abc = prod[i][j].substr(nonter[i].length(), prod[i][j].length() -
nonter[i].length()) + nonter[i] + "";

            temp.push_back(abc);

            prod[i].erase(prod[i].begin() + j);

            --j;

        }

        else

        {

            prod[i][j] += nonter[i] + "";

        }

    }

    temp.push_back("esp");

    prod.push_back(temp);

}

cout << "\n\n";

cout << "\nNew set of non-terminals: ";

for (i = 0; i < nonter.size(); ++i)

    cout << nonter[i] << " ";

cout << "\n\nNew set of productions: ";

for (i = 0; i < nonter.size(); ++i)

{

    int j;

```

```

        for (j = 0; j < prod[i].size(); ++j)
        {
            cout << "\n"
                 << nonter[i] << " -> " << prod[i][j];
        }
    }

    system("pause");

    return 0;
}

```

Output:

```

Enter number of non terminals: 3

Enter non terminals one by one:
Non terminal 1 : E

Non terminal 2 : T

Non terminal 3 : F

Enter '^' for null
Number of E productions: 2

One by one enter all E productions
RHS of production 1: E+T

RHS of production 2: T

Number of T productions: 2

One by one enter all T productions
RHS of production 1: T*F

RHS of production 2: F

Number of F productions: 2

One by one enter all F productions
RHS of production 1: (E)

RHS of production 2: i
110

New set of non-terminals: E T F E' T'

New set of productions:
E -> TE'
T -> FT'
F -> (E)
F -> i
E' -> +TE'
E' -> ^
T' -> *FT'
T' -> ^

```

b. Implementation of Left Factoring:

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
int main()
{
    system("cls");
    char ch, lhs[20][20], rhs[20][20][20], temp[20], temp1[20];
    int n, n1, count[20], x, y, i, j, k, c[20];
    printf("\nEnter the no. of nonterminals : ");
    scanf("%d", &n);
    n1 = n;
    for (i = 0; i < n; i++)
    {
        printf("\Nonterminal %d \nEnter the no. of productions : ", i + 1);
        scanf("%d", &c[i]);
        printf("\nEnter LHS : ");
        scanf("%s", lhs[i]);
        for (j = 0; j < c[i]; j++)
        {
            printf("%s->", lhs[i]);
            scanf("%s", rhs[i][j]);
        }
    }
    for (i = 0; i < n; i++)
    {
        count[i] = 1;
        while (memcmp(rhs[i][0], rhs[i][1], count[i]) == 0)
            count[i]++;
    }
}
```

```

    }
    for (i = 0; i < n; i++)
    {
        count[i]--;
        if (count[i] > 0)
        {
            strcpy(lhs[n1], lhs[i]);
            strcat(lhs[i], "");
            for (k = 0; k < count[i]; k++)
                temp1[k] = rhs[i][0][k];
            temp1[k++] = '\0';
            for (j = 0; j < c[i]; j++)
            {
                for (k = count[i], x = 0; k < strlen(rhs[i][j]); x++, k++)
                    temp[x] = rhs[i][j][k];
                temp[x++] = '\0';
                if (strlen(rhs[i][j]) == 1)
                    strcpy(rhs[n1][1], rhs[i][j]);
                strcpy(rhs[i][j], temp);
            }
            c[n1] = 2;
            strcpy(rhs[n1][0], temp1);
            strcat(rhs[n1][0], lhs[n1]);
            strcat(rhs[n1][0], "");
            n1++;
        }
    }
    printf("\n\nThe resulting productions are : \n");
    for (i = 0; i < n1; i++)
    {
        if (i == 0)
            printf("\n %s -> %c|", lhs[i], (char)238);
        else

```

```

        printf("\n %s -> ", lhs[i]);
    for (j = 0; j < c[i]; j++)
    {
        printf(" %s ", rhs[i][j]);
        if ((j + 1) != c[i])
            printf("|");
    }
    printf("\b\b\b\b\n");
}
return 0;
}

```

Output:

```

Enter the no. of nonterminals : 2

Nonterminal 1
Enter the no. of productions : 3

Enter LHS : S
S->iCtSeS
S->iCtS
S->a

Nonterminal 2
Enter the no. of productions : 1

Enter LHS : C
C->b

The resulting productions are :

S' -> ε | eS | |

C -> b

S -> iCtSS' | a

```

Result:

Elimination of ambiguity using Left Recursion and Left Factoring has been done successfully.

Experiment-5

Computation of FIRST () and FOLLOW ()

Aim:

To write a program to compute the FIRST() and FOLLOW().

Procedure:

a. For computing the first:

1. If X is a terminal then $\text{FIRST}(X) = \{X\}$

Example: $F \rightarrow I \mid id$

We can write it as $\text{FIRST}(F) \rightarrow \{ (, id)$

2. If X is a non-terminal like $E \rightarrow T$ then to get FIRSTI substitute T with other productions

until you get a terminal as the first symbol

3. If $X \rightarrow \epsilon$ then add ϵ to $\text{FIRST}(X)$.

b. For computing the follow:

1. Always check the right side of the productions for a non-terminal, whose FOLLOW set is

being found. (never see the left side).

2. (a) If that non-terminal (S,A,B...) is followed by any terminal (a,b...,*,+,(),...) , then add that terminal into the FOLLOW set.

(b) If that non-terminal is followed by any other non-terminal then add FIRST of other nonterminal into the FOLLOW set.

Code:

```
#include <bits/stdc++.h>

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

char a[10][10], f[10];

void follow(char c);

void first(char c);

int main()
{
    system("cls");

    int i, z;

    char c, ch;

    printf("Enter the no of prooductions:\n");
    scanf("%d", &n);

    printf("Enter the productions:\n");
    for (i = 0; i < n; i++)
        scanf("%s%c", a[i], &ch);
    do
    {
        m = 0;

        printf("Enter the elemets whose fisrt & follow is to be found:");
        scanf("%c", &c);

        first(c);

        printf("First(%c)={ ", c);
        for (i = 0; i < m; i++)
            printf("%c", f[i]);
        printf("}\n");
        strcpy(f, " ");
        m = 0;

        follow(c);

        printf("Follow(%c)={ ", c);
```



```

    for (i = 0; i < m; i++)
        printf("%c", f[i]);
    printf("}\n");
    printf("Continue(0/1)?");
    scanf("%d%c", &z, &ch);
} while (z == 1);
system("pause");
return (0);
}

```

```

void first(char c)
{
    int k;
    if (!isupper(c))
        f[m++] = c;
    for (k = 0; k < n; k++)
    {
        if (a[k][0] == c)
        {
            if (a[k][2] == '$')
                follow(a[k][0]);
            else if (islower(a[k][2]))
                f[m++] = a[k][2];
            else
                first(a[k][2]);
        }
    }
}

```

```

void follow(char c)
{

```

```

if (a[0][0] == c)
    f[m++] = '$';
for (i = 0; i < n; i++)
{
    for (j = 2; j < strlen(a[i]); j++)
    {
        if (a[i][j] == c)
        {
            if (a[i][j + 1] != '\0')
                first(a[i][j + 1]);
            if (a[i][j + 1] == '\0' && c != a[i][0])
                follow(a[i][0]);
        }
    }
}
}

```

Output:

```

Enter the no of prooductions:
5
Enter the productions:
S=AbCd
A=Cf
A=a
C=gE
E=h
Enter the elemets whose fisrt & follow is to be found:S
First(S)={ga}
Follow(S)={$}
Continue(0/1)?1
Enter the elemets whose fisrt & follow is to be found:A
First(A)={ga}
Follow(A)={b}
Continue(0/1)?1
Enter the elemets whose fisrt & follow is to be found:C
First(C)={g}
Follow(C)={df}

```

Result:

The FIRST and FOLLOW sets of the non-terminals of a grammar were found successfully.

Experiment-6 Implement Predictive Parsing Table

Aim:

To implement Predictive Parsing Table in C/C++.

Procedure:

1. Start the program.
2. Initialize the required variables.
3. Get the number of coordinates and productions from the user.
4. Perform the following
 - for (each production $A \rightarrow \alpha$ in G) {
 - for (each terminal a in $FIRST(\alpha)$)
 - add $A \rightarrow \alpha$ to $M[A, a]$;
 - if (ϵ is in $FIRST(\alpha)$)
 - for (each symbol b in $FOLLOW(A)$)
 - add $A \rightarrow \alpha$ to $M[A, b]$;
5. Print the resulting stack.
6. Print if the grammar is accepted or not.
7. Exit the program.

Code:

```
#include<bits/stdc++.h>

int main() {
    system("cls");

    char fin[10][20],st[10][20],ft[20][20],fol[20][20];

    int a=0,e,i,t,b,c,n,k,l=0,j,s,m,p;

    printf("Enter the no. of nonterminals\n");

    scanf("%d",&n);

    printf("Enter the productions in a grammar\n");

    for(i=0;i<n;i++)
        scanf("%s",st[i]);

    for(i=0;i<n;i++)
        fol[i][0]='\0';

    for(s=0;s<n;s++)
    {
        for(i=0;i<n;i++)
        {
            j=3;
            l=0;
            a=0;
            l1:if(!(st[i][j]>64)&&(st[i][j]<91)))
            {
                for(m=0;m<l;m++)
                {
                    if(ft[i][m]==st[i][j])
                        goto s1;
                }
                ft[i][l]=st[i][j];
                l=l+1;
                s1:j=j+1;
            }
        }
    }
}
```

```

else
{
    if(s>0)
    {
        while(st[i][j]!=st[a][0])
            a++;
        b=0;
        while(ft[a][b]!='\0')
        {
            for(m=0;m<1;m++)
            {
                if(ft[i][m]==ft[a][b])
                    goto s2;
            }
            ft[i][1]=ft[a][b];
            l=l+1;
            s2:b=b+1;
        }
    }
}
while(st[i][j]!='\0')
{
    if(st[i][j]=='|')
    {
        j=j+1;
        goto l1;
    }
    j=j+1;
}
ft[i][1]='\0';
}

```

```

}
printf("\nFirst \n");
for(i=0;i<n;i++)
    printf("FIRST[%c]=%s\n",st[i][0],ft[i]);
fol[0][0]='$';
for(i=0;i<n;i++)
{
    k=0;
    j=3;
    if(i==0)
        l=1;
    else
        l=0;
    k1:while((st[i][0]!=st[k][j])&&(k<n))
    {
        if(st[k][j]=='\0')
        {
            k++;
            j=2;
        }
        j++;
    }

    j=j+1;
    if(st[i][0]==st[k][j-1])
    {
        if((st[k][j]!='')&&(st[k][j]!='\0'))
        {
            a=0;
            if(!((st[k][j]>64)&&(st[k][j]<91)))
            {

```

```

for(m=0;m<1;m++)
{
    if(fol[i][m]==st[k][j])
        goto q3;
}
fol[i][l]=st[k][j];
l++;
q3::
}
else
{
    while(st[k][j]!=st[a][0])
    {
        a++;
    }
    p=0;
    while(ft[a][p]!='\0')
    {
        if(ft[a][p]!='@')
        {
            for(m=0;m<1;m++)
            {
                if(fol[i][m]==ft[a][p])
                    goto q2;
            }
            fol[i][l]=ft[a][p];
            l=l+1;
        }
        else
            e=1;
        q2:p++;
    }
}

```

```

    }
    if(e==1)
    {
        e=0;
        goto a1;
    }
}
else
{
    a1:c=0;
    a=0;
    while(st[k][0]!=st[a][0])
    {
        a++;
    }
    while((fol[a][c]!='\0')&&(st[a][0]!=st[i][0]))
    {
        for(m=0;m<l;m++)
        {
            if(fol[i][m]==fol[a][c])
                goto q1;
        }
        fol[i][l]=fol[a][c];
        l++;
        q1:c++;
    }
}
goto k1;
}
fol[i][l]='\0';

```



```

}
printf("\nFollow \n");
for(i=0;i<n;i++)
    printf("FOLLOW[%c]=%s\n",st[i][0],fol[i]);
printf("\n");
s=0;
for(i=0;i<n;i++)
{
    j=3;
    while(st[i][j]!='\0')
    {
        if((st[i][j-1]=='|')||(j==3))
        {
            for(p=0;p<=2;p++)
            {
                fin[s][p]=st[i][p];
            }
            t=j;
            for(p=3;((st[i][j]!='|')&&(st[i][j]!='\0')));p++)
            {
                fin[s][p]=st[i][j];
                j++;
            }
            fin[s][p]='\0';
            if(st[i][k]=='@')
            {
                b=0;
                a=0;
                while(st[a][0]!=st[i][0])
                {
                    a++;

```

```

    }
    while(fol[a][b]!='\0')
    {
        printf("M[%c,%c]=%s\n",st[i][0],fol[a][b],fin[s]);
        b++;
    }
}
else if(!((st[i][t]>64)&&(st[i][t]<91)))
    printf("M[%c,%c]=%s\n",st[i][0],st[i][t],fin[s]);
else
{
    b=0;
    a=0;
    while(st[a][0]!=st[i][3])
        a++;
    while(ft[a][b]!='\0')
    {
        printf("M[%c,%c]=%s\n",st[i][0],ft[a][b],fin[s]);
        b++;
    }
}
s++;
}
if(st[i][j]=='|')
    j++;
}
}
system("pause");
return 0;
}

```

Output:

```
Enter the no. of nonterminals
2
Enter the productions in a grammar
S->CC
C->eC|d

First
FIRST[S]=ed
FIRST[C]=ed

Follow
FOLLOW[S]=$
FOLLOW[C]=ed$

M[S,e]=S->CC
M[S,d]=S->CC
M[C,e]=C->eC
M[C,d]=C->d
```

Result:

The implementation of Predictive Parsing Table was successful.

Experiment-7 Implement Shift Reduce Parsing

Aim:

To implement Shift Reduce Parsing in C/C++.

Procedure:

Shift Reduce parser attempts for the construction of parse in a similar manner as done in bottom up parsing i.e. the parse tree is constructed from leaves(bottom) to the root(up). A more general form of shift reduce parser is LR parser. This parser requires some data structures i.e.

- A input buffer for storing the input string.
- A stack for storing and accessing the production rules.

Basic Operations –

- **Shift:** This involves moving of symbols from input buffer onto the stack.
- **Reduce:** If the handle appears on top of the stack then, its reduction by using appropriate production rule is done i.e. RHS of production rule is popped out of stack and LHS of production rule is pushed onto the stack.
- **Accept:** If only start symbol is present in the stack and the input buffer is empty then, the parsing action is called accept. When accept action is obtained, it means successful parsing is done.
- **Error:** This is the situation in which the parser can neither perform shift action nor reduce action and not even accept action.

Code:

```
#include <bits/stdc++.h>

struct prodn {
    char p1[10];
    char p2[10];
};

int main() {
    system("cls");
    char input[20], stack[50], temp[50], ch[2], *t1, *t2, *t;
    int i, j, s1, s2, s, count = 0;
    struct prodn p[10];
    FILE *fp = fopen("sr_input.txt", "r");
    stack[0] = '\0';
    printf("\n Enter the input string\n");
    scanf("%s", &input);
    while (!feof(fp)) {
        fscanf(fp, "%s\n", temp);
        t1 = strtok(temp, "->");
        t2 = strtok(NULL, "->");
        strcpy(p[count].p1, t1);
        strcpy(p[count].p2, t2);
        count++;
    }
    i = 0;
    while (1) {
        if (i < strlen(input))
        {
            ch[0] = input[i];
            ch[1] = '\0';
            i++;
            strcat(stack, ch);
        }
    }
}
```

```

        printf("%s\n", stack);
    }
    for (j = 0; j < count; j++) {
        t = strstr(stack, p[j].p2);
        if (t != NULL)
        {
            s1 = strlen(stack);
            s2 = strlen(t);
            s = s1 - s2;
            stack[s] = '\0';
            strcat(stack, p[j].p1);
            printf("%s\n", stack);
            j = -1;
        }
    }
    if (strcmp(stack, "E") == 0 && i == strlen(input))
    {
        printf("\n Accepted\n");
        break;
    }
    if (i == strlen(input))
    {
        printf("\n Not Accepted\n");
        break;
    }
}
system("pause");
return 0;
}

```

Input File:

$E \rightarrow E + E$
 $E \rightarrow E * E$
 $E \rightarrow i$

Output 1:

```
Enter the input string
i*i+i
i
E
E*
E*i
E*E
E
E+
E+i
E+E
E

Accepted
```

Output 2:

```
Enter the input string
i*i+i
i
E
E*
E*+
E*+i
E*+E

Not Accepted
```

Result:

The implementation of Shift Reduce Parsing was successful.

Experiment-8 Implement LEADING AND TRAILING

Aim:

To implement Leading and Trailing for the given grammar in C/C++.

Procedure:

- 1. For Leading, check for the first non-terminal.**
- 2. If found, print it.**
- 3. Look for next production for the same non-terminal.**
- 4. If not found, recursively call the procedure for the single non-terminal present before
the
comma or End of Production String.**
- 5. Include its results in the result of this non-terminal.**
- 6. For trailing, we compute same as leading but we start from the end of the
production to
the beginning.**

Code:

```
#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 << " : ";
```

```
        cin >> gram[i].prodno;
```

```
        for (j = 0; j < gram[i].prodno; j++)
```

```
        {
```

```

        cout << gram[i].lhs << "->";
        cin >> gram[i].rhs[j];
    }
}
}

```

```

void leading() {
    for (i = 0; i < vars; i++)
    {
        for (j = 0; j < gram[i].prodno; j++)
        {
            for (k = 0; k < terms; k++)
            {
                if (gram[i].rhs[j][0] == term[k])
                    lead[i][k] = 1;
                else
                {
                    if (gram[i].rhs[j][1] == term[k])
                        lead[i][k] = 1;
                }
            }
        }
    }
    for (rep = 0; rep < vars; rep++)
    {
        for (i = 0; i < vars; i++)
        {
            for (j = 0; j < gram[i].prodno; j++)
            {
                for (m = 1; m < vars; m++)
                {

```

```

        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] != '\x00')
                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() {
    system("cls");
    get();
    leading();
    trailing();
    display();
    cout << "\n";
    system("pause");
    return 0;
}

```

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 implementation of Leading and Trailing was successful.

Experiment-9 Implement LR(0) Items

Aim:

To implement LR(0) items for the given grammar in C/C++.

Procedure:

1. Start.
2. Create structure for production with LHS and RHS.
3. Open file and read input from file.
4. Build state 0 from extra grammar Law $S' \rightarrow S \$$ that is all start symbol of grammar and one

Dot (.) before S symbol.

5. If Dot symbol is before a non-terminal, add grammar laws that this non-terminal is in Left

Hand Side of that Law and set Dot in before of first part of Right Hand Side.

6. If state exists (a state with this Laws and same Dot position), use that instead.
7. Now find set of terminals and non-terminals in which Dot exist in before.
8. If step 7 Set is non-empty go to 9, else go to 10.

9. For each terminal/non-terminal in set step 7 create new state by using all grammar law that

Dot position is before of that terminal/non-terminal in reference state by increasing Dot point

to next part in Right Hand Side of that laws.

10. Go to step 5.

11. End of state building.

12. Display the output.

13. End.

Code:

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

char prod[20][20],listofvar[26]="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
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;
    }
}
}
}
}
arr[noitem]=n;
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 augmented grammar \n";
for(i=0;i<noavar;i++)
    cout<<endl<<g[i].lhs<<"->"<<g[i].rhs<<" ";

for(i=0;i<noavar;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++]=noavar;
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<1;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(Ø TO END) :

```

```

E->E+T

```

```

E->T

```

```

T->T*F

```

```

T->F

```

```

F->E

```

```

F->i

```

```

Ø

```

```

augumented grammar

```

```

A->E

```

```

E->E+T

```

```

E->T

```

```

T->T*F

```

```

T->F

```

```

F->E

```

```

F->i

```

```

THE SET OF ITEMS ARE

```

```

IØ

```

```

A->.,E

```

```

E->.,E+T

```

```

E->.,T

```

```

T->.,T*F

```

```

T->.,F

```

```

F->.,E

```

```

F->.,i

```

```

I1

```

```

A->E.

```

```

E->E.+T

```

```

F->E.

```

```

I2

```

```

E->T.

```

```

T->T.*F

```

```

I3

```

```

T->F.

```

I4

F→i.

I5

E→E+.T

T→.T*F

T→.F

F→.E

F→.i

E→.E+T

E→.T

I6

T→T*.F

F→.E

F→.i

E→.E+T

E→.T

T→.T*F

T→.F

I7

E→E+T.

T→T.*F

E→T.

I8

F→E.

E→E.+T

I9

T→T*F.

T→F.

Result:

The implementation of LR(0) items was successful.

Experiment-10

Intermediate Code Generation – Postfix, Prefix

Aim:

To implement intermediate code generation in C/C++.

Procedure:

1. Declare set of operators.
2. Initialize an empty stack.
3. To convert INFIX to POSTFIX follow the following steps
4. Scan the infix expression from left to right.
5. If the scanned character is an operand, output it.
6. Else, If the precedence of the scanned operator is greater than the precedence of the operator in the stack(or the stack is empty or the stack contains a '('), push it.
7. Else, Pop all the operators from the stack which are greater than or equal to in precedence
than that of the scanned operator. After doing that Push the scanned operator to the stack.
8. If the scanned character is an '(', push it to the stack.
9. If the scanned character is an ')', pop the stack and output it until a '(' is encountered, and
discard both the parenthesis.
10. Pop and output from the stack until it is not empty.
11. To convert INFIX to PREFIX follow the following steps

12. First, reverse the infix expression given in the problem.
13. Scan the expression from left to right.
14. Whenever the operands arrive, print them.
15. If the operator arrives and the stack is found to be empty, then simply push the operator into the stack.
16. Repeat steps 6 to 9 until the stack is empty

Code:

Infix to Postfix:

```
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<string.h>
#define SIZE 100
char stack[SIZE];
int top = -1;
void push(char item) {
    if(top >= SIZE-1)
        printf("\nStack Overflow.");
    else {
        top = top+1;
        stack[top] = item;
    }
}
char pop() {
    char item ;
    if(top <0) {
        printf("stack under flow: invalid infix expression");
        getchar();
        exit(1);
    }
}
```



```

    }
    else {
        item = stack[top];
        top = top-1;
        return(item);
    }
}

int is_operator(char symbol) {
    if(symbol == '^' || symbol == '*' || symbol == '/' || symbol == '+' || symbol == '-')
        return 1;
    else {
        return 0;
    }
}

int precedence(char symbol) {
    if(symbol == '^')
        return(3);
    else if(symbol == '*' || symbol == '/')
        return(2);
    else if(symbol == '+' || symbol == '-')
        return(1);
    else
        return(0);
}

void InfixToPostfix(char infix_exp[], char postfix_exp[]) {
    int i, j;
    char item;
    char x;
    push('(');
    strcat(infix_exp, "");

```

```

i=0; j=0;
item=infix_exp[i];
while(item != '\0') {
    if(item == '(')
        push(item);
    else if( isdigit(item) || isalpha(item)) {
        postfix_exp[j] = item;
        j++;
    }
    else if(is_operator(item) == 1) {
        x=pop();
        while(is_operator(x) == 1 && precedence(x)>= precedence(item)) {
            postfix_exp[j] = x;
            j++;
            x = pop();
        }
        push(x);
        push(item);
    }
    else if(item == ')') {
        x = pop();
        while(x != '(') {
            postfix_exp[j] = x;
            j++;
            x = pop();
        }
    }
    Else {
        printf("\nInvalid infix Expression.\n");
        getchar();
    }
}

```

```

        exit(1);
    }
    i++;
    item = infix_exp[i];
}
if(top>0) {
    printf("\nInvalid infix Expression.\n");
    getchar();
    exit(1);
}
if(top>0) {
    printf("\nInvalid infix Expression.\n");
    getchar();
    exit(1);
}
postfix_exp[j] = '\0';
}

int main() {
    char infix[SIZE], postfix[SIZE];

    printf("ASSUMPTION: The infix expression contains single letter variables and single digit constants only.\n");

    printf("\nEnter Infix expression : ");
    gets(infix);
    InfixToPostfix(infix,postfix);
    printf("Postfix Expression: ");
    puts(postfix);
    printf("\n");
    return 0;
}

```

Postfix to Prefix:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
#define MAX 20
char str[MAX], stack[MAX];
int top = -1;
void push(char c) { stack[++top] = c; }
char pop(){ return stack[top--];}
int checkIfOperand(char ch){ return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z');}

int isOperator(char x) {
    switch (x) {
        case '+':
        case '-':
        case '/':
        case '*':
            return 1;
        }
    return 0;
}

void postfixToPrefix() {
    int n, i, j = 0;
    char c[20];
    char a, b, op;
    printf("Enter the postfix expression:\n");
    scanf("%s", str);
    n = strlen(str);
    for (i = 0; i < MAX; i++)
```

```

    stack[i] = '\0';
printf("Prefix expression is:\t");
for (i = n - 1; i >= 0; i--) {
    if (isOperator(str[i])) push(str[i]);
    else { c[j++] = str[i];
        while ((top != -1) && (stack[top] == '#')) {
            a = pop();
            c[j++] = pop();
        }
        push('#');
    }
}
c[j] = '\0';
i = 0;
j = strlen(c) - 1;
char d[20];
while (c[i] != '\0') {
    d[j--] = c[i++];
}
printf("%s\n", d);
}

int main() {
    postfixToprefix();
    return 0;
}

```

Output:

```
INPUT THE EXPRESSION: A+B^C/R  
PREFIX: +^/CR  
POSTFIX: AB^CR/+
```

Result:

The implementation of intermediate code generation was successful.

Experiment-11 Three Address Code Generation

Aim:

To implement three address code generation in C/C++.

Procedure:

1. Invoke a function getreg to find out the location L where the result of computation $b \text{ op } c$ should be stored.
2. Consult the address description for y to determine y'. If the value of y currently in memory and register both then prefer the register y'. If the value of y is not already in L then generate the instruction $\text{MOV } y', L$ to place a copy of y in L.
3. Generate the instruction $\text{OP } z', L$ where z' is used to show the current location of z. if z is in both then prefer a register to a memory location. Update the address descriptor of x to indicate that x is in location L. If x is in L then update its descriptor and remove x from all other descriptors.
4. If the current value of y or z have no next uses or not live on exit from the block or in register then alter the register descriptor to indicate that after execution of $x := y \text{ op } z$ those register will no longer contain y or z.

Code:

```
#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:

```

Enter the expression:a=b+c-d
The Intermediate code is:
t1=b+c
t2=t1-d
a=t2

```

Result:

The implementation of three address code was successful.

Experiment-12

Simple Code Generator

Aim:

To implement simple code generator in C/C++.

Procedure:

1. Parse the input code and generate an abstract syntax tree (AST).
2. Traverse the AST and generate intermediate code (e.g. three-address code).
3. Optimize the intermediate code to improve performance.
4. Generate target code (e.g. machine code) from the optimized intermediate code.
5. Output the generated code to a file or execute it directly.

Code:

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cctype>
using namespace std;
```

```
typedef struct
```

```
{  
    char var[10];  
    int alive;  
} regist;
```

```
regist preg[10];
```

```
void substring(char exp[], int st, int end)
```

```
{  
    int i, j = 0;  
    char dup[10] = "";  
    for (i = st; i < end; i++)  
        dup[j++] = exp[i];  
    dup[j] = '0';  
    strcpy(exp, dup);  
}
```

```
int getregister(char var[])
```

```
{  
    int i;  
    for (i = 0; i < 10; i++)  
    {  
        if (preg[i].alive == 0)  
        {  
            strcpy(preg[i].var, var);  
            break;  
        }  
    }
```

```

    }
    return (i);
}

```

```

void getvar(char exp[], char v[])
{
    int i, j = 0;
    char var[10] = "";
    for (i = 0; exp[i] != '\0'; i++)
        if (isalpha(exp[i]))
            var[j++] = exp[i];
        else
            break;
    strcpy(v, var);
}

```

```

int main()
{
    char basic[10][10], var[10][10], fstr[10], op;
    int i, j, k, reg, vc, flag = 0;
    cout << "\nEnter the Three Address Code:\n";
    for (i = 0;; i++)
    {
        cin.getline(basic[i], 10);
        if (strcmp(basic[i], "exit") == 0)
            break;
    }
    cout << "\nThe Equivalent Assembly Code is:\n";
}

```

```

for (j = 0; j < i; j++)
{
    getvar(basic[j], var[vc++]);
    strcpy(fstr, var[vc - 1]);
    substring(basic[j], strlen(var[vc - 1]) + 1, strlen(basic[j]));
    getvar(basic[j], var[vc++]);
    reg = getregister(var[vc - 1]);
    if (preg[reg].alive == 0)
    {
        printf("\nMov R%d,%s", reg, var[vc - 1]);
        preg[reg].alive = 1;
    }
    op = basic[j][strlen(var[vc - 1])];
    substring(basic[j], strlen(var[vc - 1]) + 1, strlen(basic[j]));
    getvar(basic[j], var[vc++]);
    switch (op)
    {
    case '+':
        cout << "\nAdd";
        break;
    case '-':
        cout << "\nSub";
        break;
    case '*':
        cout << "\nMul";
        break;
    case '/':
        cout << "\nDiv";

```

```

        break;
    }
    flag = 1;
    for (k = 0; k <= reg; k++)
    {
        if (strcmp(preg[k].var, var[vc - 1]) == 0)
        {
            cout << "R" << k << ", R" << reg;
            preg[k].alive = 0;
            flag = 0;
            break;
        }
    }
    if (flag)
    {
        printf(" %s,R%d", var[vc - 1], reg);
        printf("\nMov %s,R%d", fstr, reg);
    }
    strcpy(preg[reg].var, var[vc - 3]);
}
return 0;
}

```

Output:

```
Enter the Three Address Code:
```

```
a=b+c
```

```
c=a*c
```

```
exit
```

```
The Equivalent Assembly Code is:
```

```
Mov R0,b
```

```
Add c,R0
```

```
Mov a,R0
```

```
Mov R1,a
```

```
Mul c,R1
```

```
Mov c,R1
```

Result:

The implementation of simple code generator was successful.

Experiment-13

Construction of DAG

Aim:

To implement the construction of DAG using C/ C++.

Procedure:

1. Start the program.
2. Include all the header files.
3. Check for postfix expression and construct the in-order DAG representation.
4. Print the output.
5. Stop the program.

Code:

```
#include <iostream>
#include <string>
#include <unordered_map>
using namespace std;
class DAG
{
public:
    char label;
    char data;
    DAG *left;
```

```
DAG *right;
```

```
DAG(char x)
```

```
{  
    label = '_';  
    data = x;  
    left = NULL;  
    right = NULL;  
}
```

```
DAG(char lb, char x, DAG *lt, DAG *rt)
```

```
{  
    label = lb;  
    data = x;  
    left = lt;  
    right = rt;  
}  
};
```

```
int main()
```

```
{  
    int n;  
    n = 3;  
    string st[n];  
    st[0] = "A=x+y";  
    st[1] = "B=A*z";  
    st[2] = "C=B/x";  
    unordered_map<char, DAG *> labelDAGNode;
```

```
for (int i = 0; i < 3; i++)
```

```
{  
    string stTemp = st[i];
```

```

for (int j = 0; j < 5; j++)
{
    char tempLabel = stTemp[0];
    char tempLeft = stTemp[2];
    char tempData = stTemp[3];
    char tempRight = stTemp[4];
    DAG *leftPtr;
    DAG *rightPtr;
    if (labelDAGNode.count(tempLeft) == 0)
    {
        leftPtr = new DAG(tempLeft);
    }
    else
    {
        leftPtr = labelDAGNode[tempLeft];
    }
    if (labelDAGNode.count(tempRight) == 0)
    {
        rightPtr = new DAG(tempRight);
    }
    else
    {
        rightPtr = labelDAGNode[tempRight];
    }
    DAG *nn = new DAG(tempLabel, tempData, leftPtr, rightPtr);
    labelDAGNode.insert(make_pair(tempLabel, nn));
}
}

cout << "Label    ptr    leftPtr    rightPtr" << endl;
for (int i = 0; i < n; i++)
{

```

```

DAG *x = labelDAGNode[st[i][0]];
cout << st[i][0] << "      " << x->data << "      ";
if (x->left->label == '_')
    cout << x->left->data;
else
    cout << x->left->label;
cout << "      ";
if (x->right->label == '_')
    cout << x->right->data;
else
    cout << x->right->label;
cout << endl;
}
return 0;
}

```

Output:

Label	ptr	leftPtr	rightPtr
A	+	x	y
B	*	A	z
C	/	B	x

Result:

The construction of DAG was successful.