Name: **Devansh Pareek**

Register Numbe **RA2011003010184**

Course Code: **18CSC304J**

Course Name: **Compiler Design**

Date of Submission: **14/05/23**

TABLE OF CONTENTS

ASSIGNMENT 1

18CSC304J_COMPILER DESIGN

UNIT I

Correct! Well done.
Your score is 100%.

Across: 17: Parsers are expected to parse the whole code? TRUE Enter

Down: 17: How many parts of compiler are there? TWO Enter

1R	E	G	U	L	2A	R	E	X	P	R	E	S	S	I	O	N								
					S																			
					S																			
				3L	E	X	I	C	A	L	A	N	A	L	Y	S	I	S						
					M																			
					B																			
			4F	I	L	E	I	N	C	L	U	S	I	O	N									
					Y												5S	O	U	R	C	E	C	O	D	E
					L												Y								
				6M	A	C	R	O	E	X	P	A	N	S	I	O	N								
					N												T								
					G			7T								8H	A	S	H	T	A	B	L	E	
			9S	O	U	R	C	E	P	R	O	G	R	A	M		X								
					A			N									E								
		10T			G										11P	U	R	E	H	L	L			
		12O	B	J	E	C	T	13P	R	O	G	R	A	M			R								
		K						A								14C	O	M	P	I	L	E	R		
	15S	E	C	O	N	D		R									R								
		N						S																	
16P	A	S	S	1				E																	

							[17]T	R	U	E															
							W																		
							O																		

Check

Across:

1. Input of lex is set of _____.
3. In a compiler, keywords of a language are recognized during the_____of the program
4. Role of preprocessor
5. which is the permanent data base in the geneal model of Compiler ?
6. Function of preprocessor
8. The symbol table implementation is based on the property of locality of reference is_____.
9. A compiler program written in a high level language is called_____.
11. Preprocessor converts high level language to_____
12. _____is a Translation of high-level language into machine language
14. By whom is the symbol table created?
15. Which phase of compiler is Syntax Analysis?
16. When is the symbol table created?
17. Parsers are expected to parse the whole code?

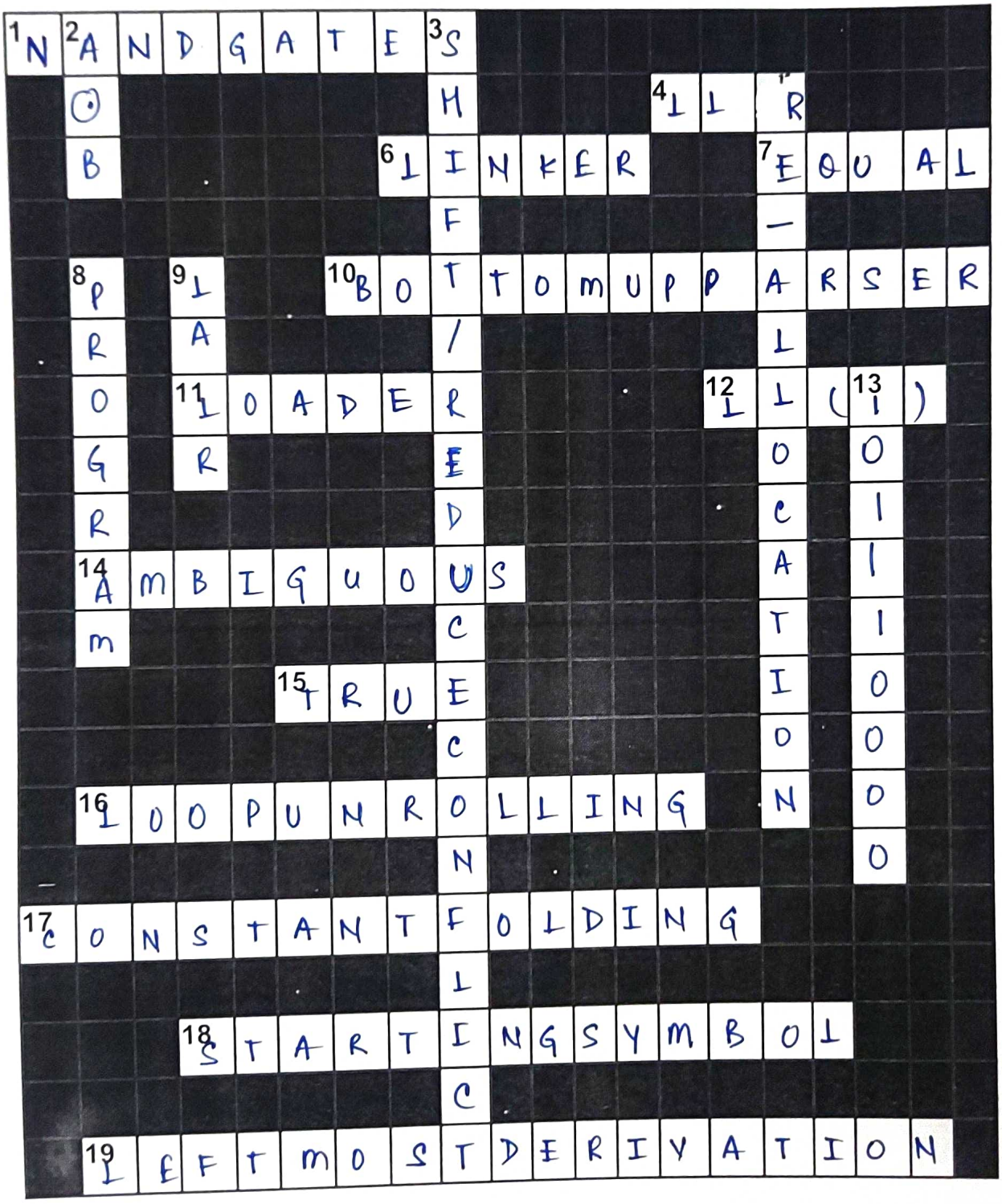
Down:

2. Assembler converts _____to machine language,
5. Compiler can check
7. The number of tokens in the following C statement is printf("i = %d, &i = %x", i, &i);
10. The output of lexical analyzer is set of _____.
13. Which concept of grammar is used in the compiler?
17. How many parts of compiler are there?

ASSIGNMENT 2

18CSC304J/Compiler Design/Dr.Shiny Irene D

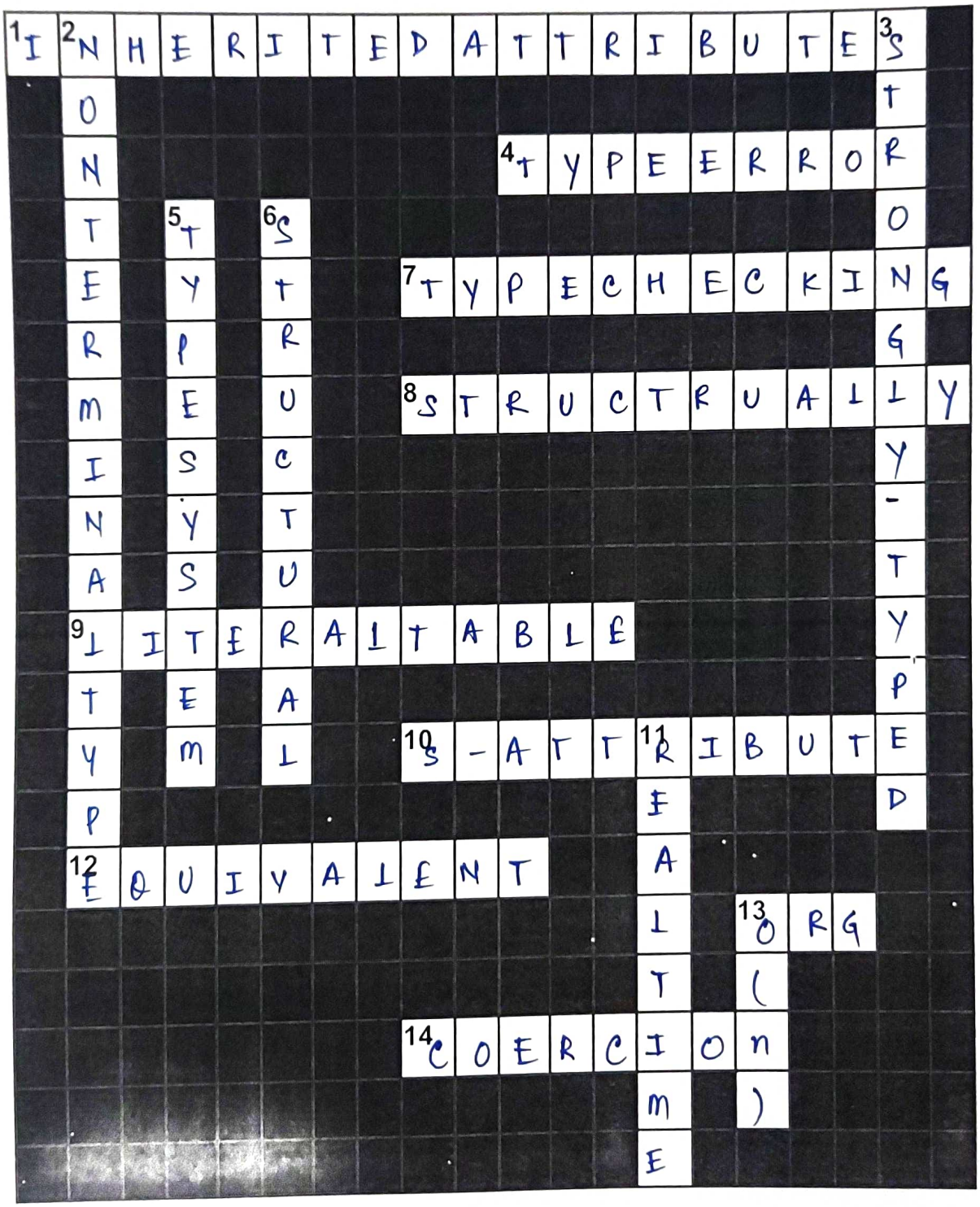
UNIT II



ASSIGNMENT 3

18CSC304J/Compiler Design/Dr.Shiny Irene D

UNIT III

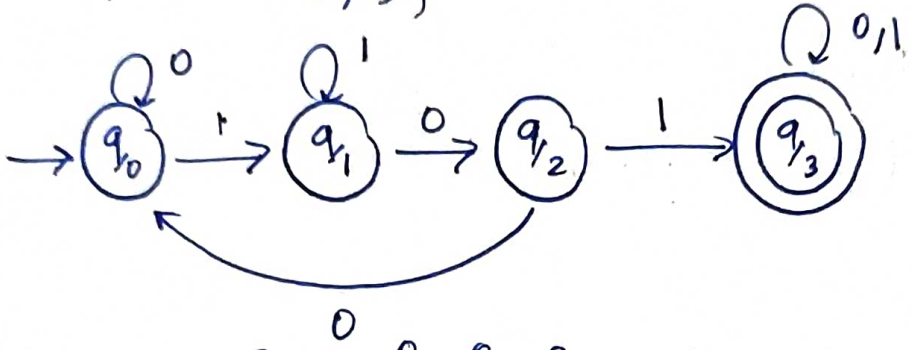


ASSIGNMENT 4

Compiler Design

Q1. The DFA for accepting strings having 101 as substring is as follows:-

$Q = \{q_0, q_1, q_2, q_3\}$



Formal Definition is given in the form

$DFA = \{Q, \Sigma, \delta, q_0, F\}$

$= \{\{q_0, q_1, q_2, q_3\}, \{0,1\}, \delta, q_0, q_3\}$

Transition table :-

State	0	1
q_0	q_0	q_1
q_1	q_2	q_1
q_2	q_0	q_3
q_3	q_3	q_3

Input string = 110100

$\delta(q_0, 110100) = \delta(q_1, 10100)$

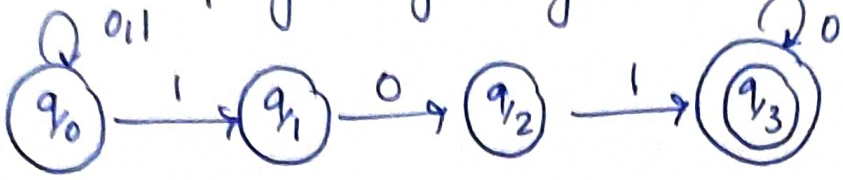
$= \delta(q_1, 0100)$

$= \delta(q_2, 100)$

$= \delta(q_3, 00)$

$= \delta(q_3, 0) = q_3 \rightarrow$ Final state. string is accepted.

② The NFA for accepting string having 101 as substring:



Formal definition:-

$NFA = \{Q, \Sigma, \delta, q_0, F\}$

$= \{\{q_0, q_1, q_2, q_3\}, \{0,1\}, \delta, q_0, q_3\}$

Transition Table:-

State	0	1
q_0	q_0	$\{q_0, q_1\}$
q_1	q_2	-
q_2	-	q_3
q_3	q_3	q_3

Input = 110100

$\delta(q_0, 110100) = \delta(\{q_0, q_1\}, 10100)$

$= \delta(\{q_0, q_1\}, 0100)$

$= \delta(\{q_0, q_2\}, 100)$

$= \delta(\{q_0, q_1, q_3\}, 00)$

$= \delta(\{q_0, q_2, q_3\}, 0)$

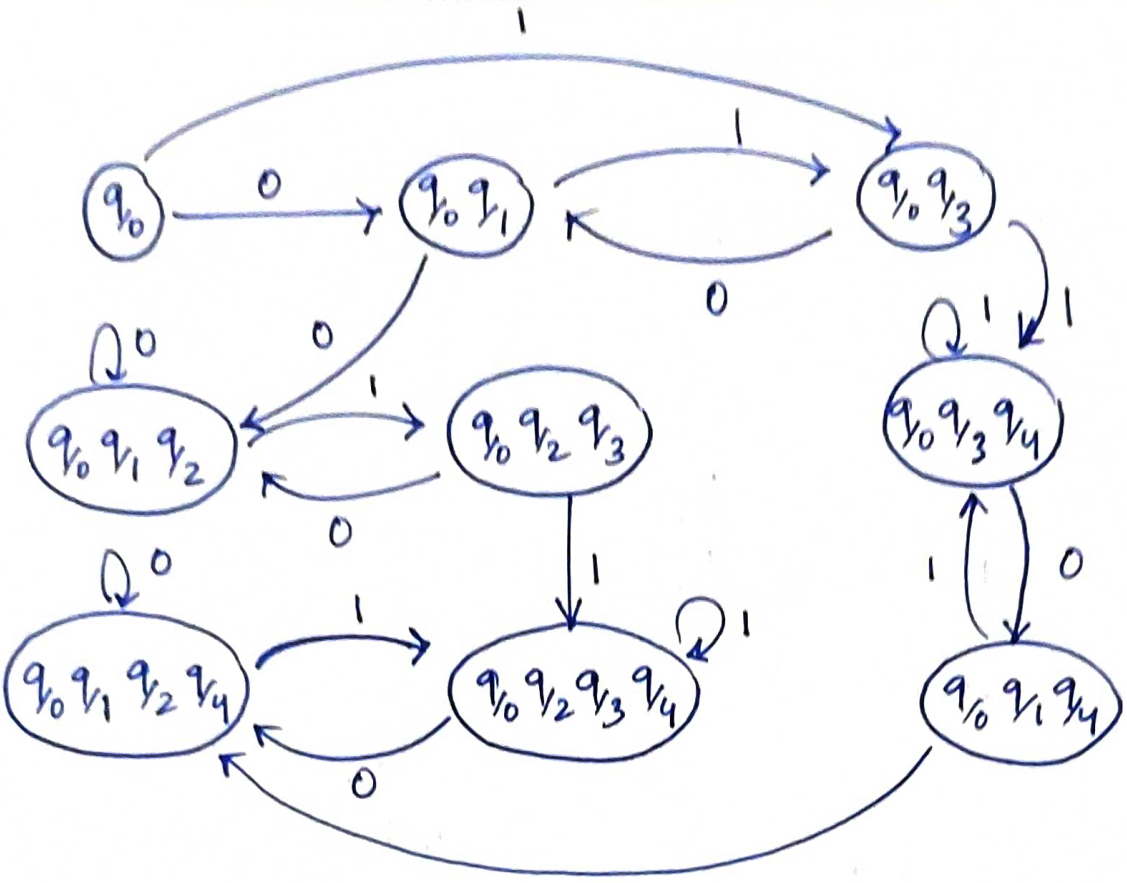
q_0, q_3 are the possible states for given string. Since q_3 is the final state, the string is accepted.

(3) Transition table for given NFA is:-

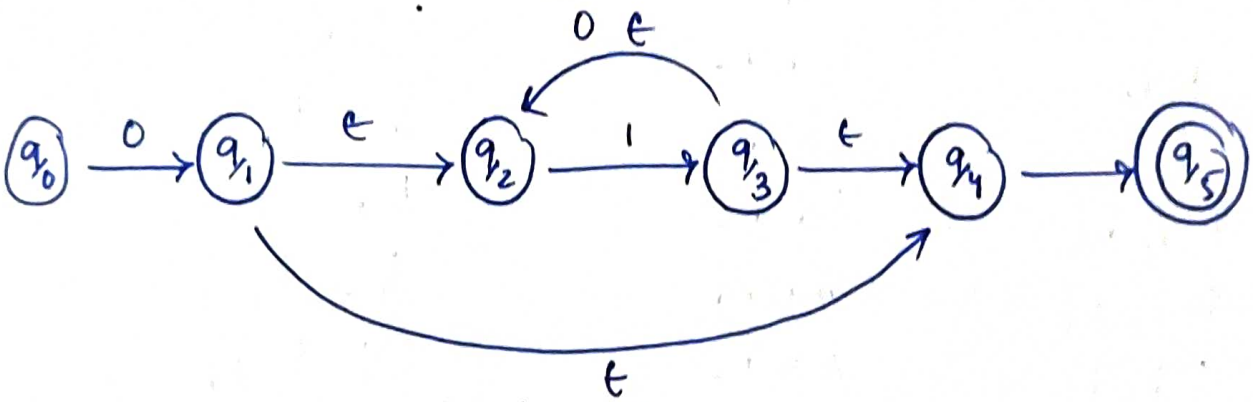
State	0	1
q_0	$q_0 q_1$	$q_0 q_3$
q_1	q_2	–
q_2	q_2	q_2
q_3	–	q_4
q_4	q_4	q_4

The equivalent DFA table is:-

State	0	1
q_0	$q_0 q_1$	$q_0 q_3$
$q_0 q_1$	$q_0 q_1 q_2$	$q_0 q_3$
$q_0 q_3$	$q_0 q_1$	$q_0 q_3 q_4$
$q_0 q_1 q_2$	$q_0 q_1 q_2$	$q_0 q_2 q_3$
$q_0 q_3 q_4$	$q_0 q_1 q_4$	$q_0 q_3 q_4$
$q_0 q_2 q_3$	$q_0 q_1 q_2$	$q_0 q_2 q_3 q_4$
$q_0 q_1 q_4$	$q_0 q_1 q_2 q_4$	$q_0 q_3 q_4$
$q_0 q_2 q_3 q_4$	$q_0 q_1 q_2 q_4$	$q_0 q_2 q_3 q_4$
$q_0 q_1 q_2 q_4$	$q_0 q_1 q_2 q_4$	$q_0 q_2 q_3 q_4$



4.



ε-closure $\{q_0\} = \{q_0\}$ — (A)

D-transition $\{A, 0\} = \{q_1\}$

ε-closure $\{q_1\} = \{q_1, q_2, q_4\}$ — (B)

D-trans $\{A, 1\} = -$

D-trans $\{B, 0\} = \{q_5\}$

ε-closure $\{q_5\} = \{q_5\}$ — (C)

D trans $(B, 1) = \{q_3\}$

ε-closure $\{q_3\} = \{q_2, q_3, q_4\}$ — (D)

D-trans (C, 0) = -

D-trans (C, 1) = -

D-trans (D, 0) = {q_5}

ε-closure{q_5} = {q_5} — (C)

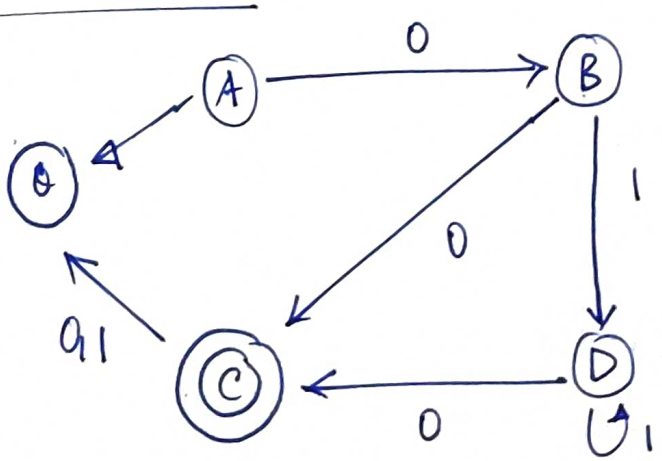
D-trans (D, 1) = {q_5}

ε-closure{q_3} = {q_2, q_3, q_4} — (D)

Equivalent table is :-

NFA	DFA	a	b
q_0	A	B	-
q_1, q_2, q_4	B	C	D
q_5	C	-	-
q_2, q_3, q_4	D	C	D

Equivalent DFA is :-

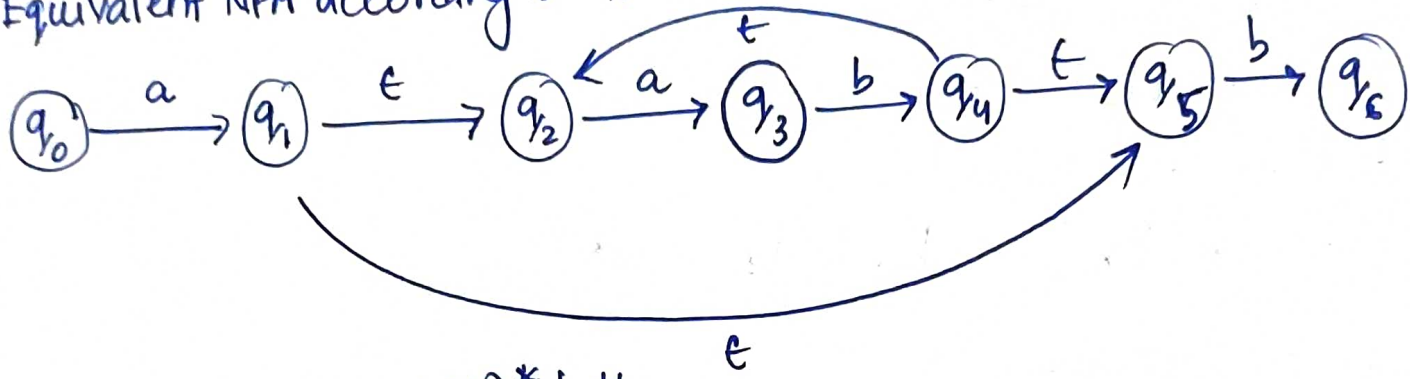


Q is the dead state

5. Given

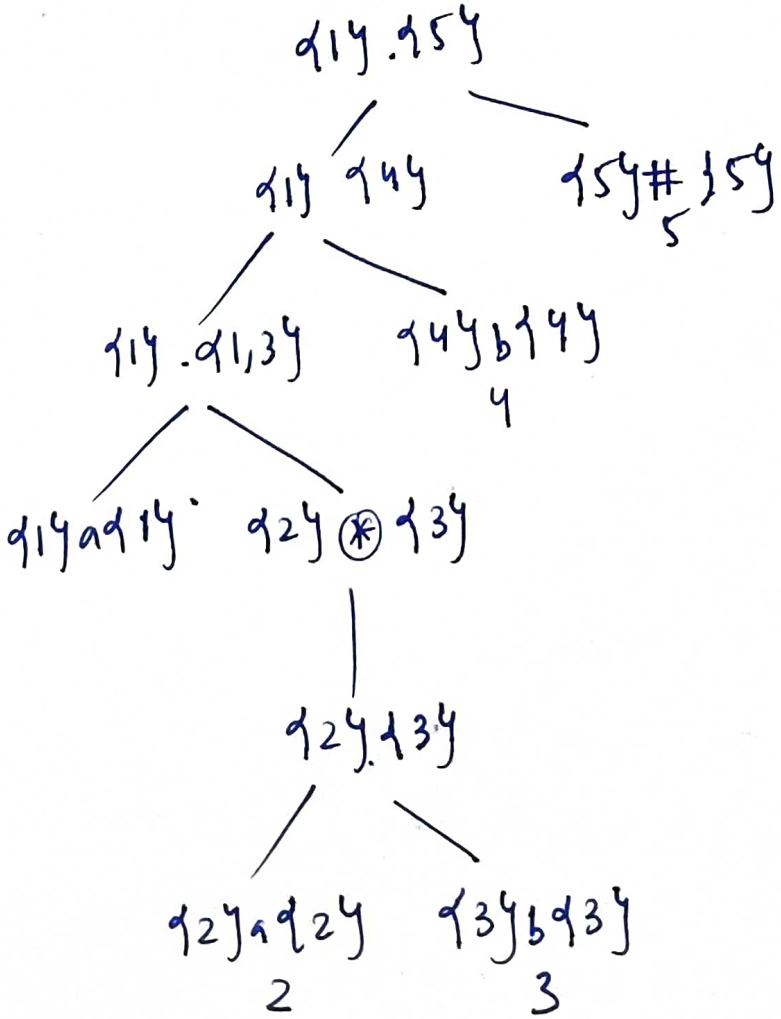
$a(ab)^*b$

Equivalent NFA according to Thomson method:

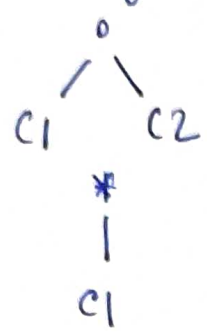


Augmented RE : $a(ab)^*b\#$

syntax tree



* is only nullable node following the rules. Now, we use following rule to find first pos & last pos:-

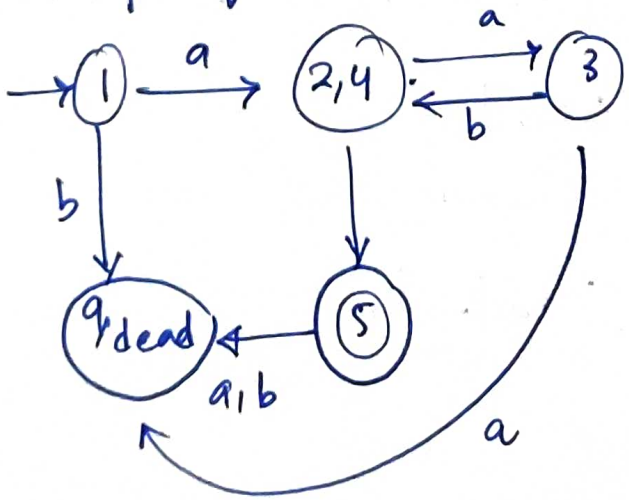
Node(n)	firstpos(n)	lastpos(n)
leaf i	{i}	{i}
	if null(c1) {fp(c1) ∪ fp(c2)} else fp(c1)	if null(c2) {lp(c1) ∪ lp(c2)} else lp(c1)
	fp(c1)	lp(c2)

Now using rules to find pos, the table is

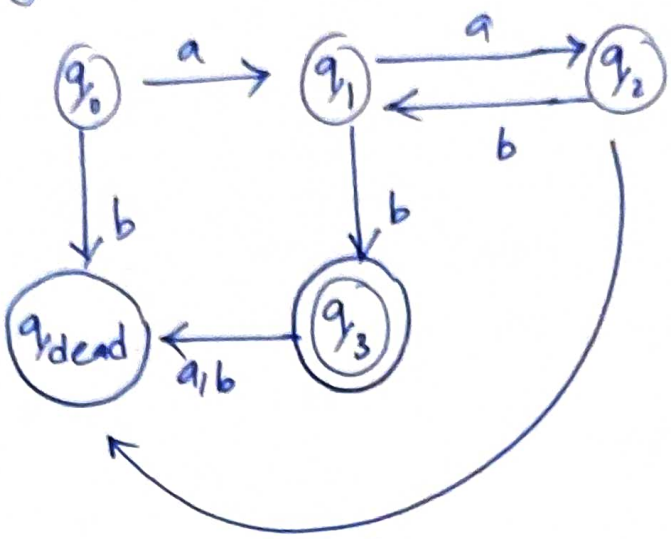
	NODE	Follow pos
a	1	2,4
a	2	3
b	3	2,4
b	4	5
#	5	–

Now constructing DFA

First pos of root node is the initial state



Renaming the states we get our final DFA



(6) Transition table for given is :-

State	0	1
q_0	q_3	q_1
q_1	q_2	q_5
q_2	q_2	q_5
q_3	q_0	q_4
q_4	q_2	q_5
q_5	q_5	q_5

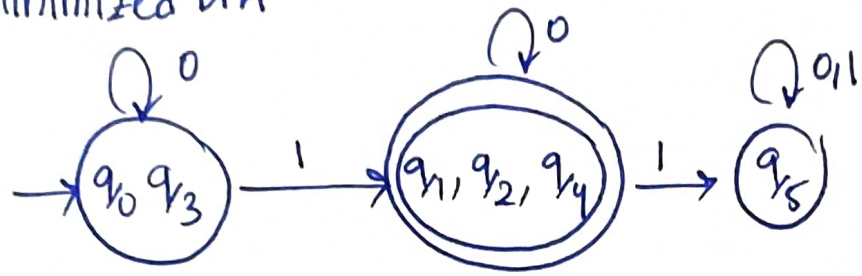
By using set method seperating all the final state we get,

$$\Pi_0 = \{q_0, q_3, q_5\}, \{q_1, q_2, q_4\}$$

q_5 have self loop while other element of its doesn't

$$\Pi_1 = \{q_0, q_3\} \{q_5\} \{q_1, q_2, q_4\}$$

Minimized DFA



minimized transition table :-

state	0	1
$q_1 q_2 q_4$	$q_1 q_2 q_4$	q_5
$q_0 q_3$	$q_0 q_3$	$q_1 q_2 q_4$
q_5	q_5	q_5

EXPERIMENTS

(1 to 13)

Experiment-1. Date:- 18/01/23

Write a program to implement lexical analyzer in c/c++.

Aim:- Write a program to implement lexical analyzer.

Algorithm:-

* Open the program file.
* If file has been opened successfully then :-
* while it is not the end of file do :-
    * Parse the file line by line
    * Check if the word in buffer currently is
        - keyword
        - operator
        - Identifier
    * Inverse the count of corresponding token.
* close the file
* output the number of :-
    - keywords
    - operators
    - Identifiers.

Code:-

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
int iskeyword(char buffer[]){
 char keywords[32][10] = {"auto","break","case","char","const",
 "continue","default","if","int","do","double","else","enum",
 "extern","float","for","goto","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(keyword[i], buffer)==0){
flag=1;
break;
}}
```

```
return flag;
}

int main()
{
  char ch, buffer[15], operators[] = "+-*/%=";
  FILE *fp;
  int i,j=0;
  fp=fopen("program.txt","r");
  if(fp==NULL){
    printf("error while opening the file\n");
    exit(0);
  }
  while((ch=fgetc(fp))!=EOF){
    for(i=0;i<6;++i){
      if(ch==operators[i])
      printf("%c is operator\n",ch);
    }
    if(isalnum(ch)){
      buffer[j++]=ch;
    }
    else if((ch==' '||ch=='\n')&&(j!=0)){
      buffer[j]='\0';
      j=0;
      if(iskeyword(buffer)==1)
      printf("%s is keyword\n",buffer);
      else
      printf("%s is identifier\n",buffer);
    }
}
fclose(fp);
return 0;
}
```

Result:- Program Executed successfully Thus, Lexical Analyzer code implemented & runs successfully

SAMPLE INPUT:- program.txt

SAMPLE OUTPUT :-

No. of keywords : 8
No. of identifier : 39
No. of operators : 40.

Verified

Experiment-2 Date:- 02/02/23.

Write a program to construct a Non-Deterministic Finite Automata (NFA) from Regular Expression (RE) using c/c++.

Aim:- WAP to convert RE to NFA.

Algorithm :-

* Include the necessary header files.
* Input the RE as a string
* Declare the functions with input conditions.
* Start the while loop.
* Initiate the while loop, check for conditions based on indices of the string & create the necessary transitions.
* Print the transition table

Program :-

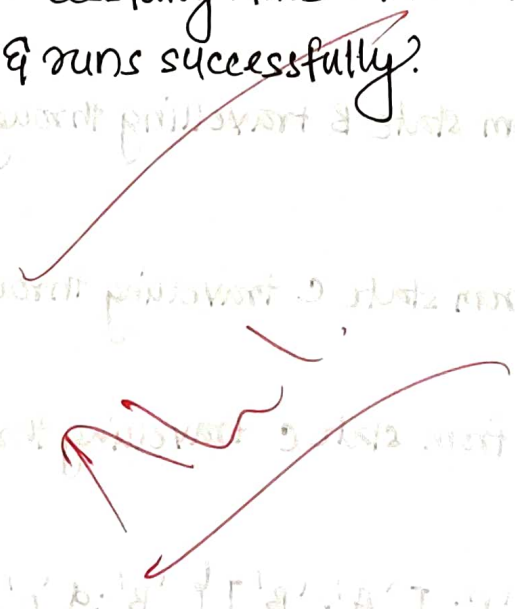
```
#include<stdio.h>
#include<string.h>
int main()
{ char reg[20]; int q[20][3], i=0, j=1, len, a, b;
  for(a=0; a<20; a++) for(b=0; b<3; b++) q[a][b]=0;
  scanf("%s", reg);
  printf("Given regular expression : %s\n", reg);
  len=strlen(reg);
  while(i<len)
  {
    if(reg[i]=='0' && reg[i+1]!='1' && reg[i+1]!='*')
        { q[j][0]=j+1; j++; }
    if(reg[i]=='b' && reg[i+1]!='1' && reg[i+1]!='*')
        { q[j][1]=j+1; j++; }
    if(reg[i]=='c' && reg[i+1]!='1' && reg[i+1]!='*')
        { q[j][2]=j+1; j++; }
    if(reg[i]=='a' && reg[i+1]=='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][1]= j+1 ; j++;
    q[j][2]= j+1 ; j++;
    i=i+2;
}
if(reg[i]=='a' && reg[i+1]=='*'){
    q[j][2]= ((j+1)*10)+(j+3); j++;
    q[j][0]= j+1 ; j++;
    q[j][2]= (j+1)*10)+(j-1); j++; }
if(reg[i]=='b'&& reg[i+1]=='*'){
    q[j][2]=((j+1)*10)+(j+3); j++;
    q[j][1] = j+1; j++;
    q[j][2]= ((j+1)*10)+(j-1); j++; }
if(reg[i]==')'&& reg[i+1]=='*'){
    q[0][2]=((j+1)*10)+1;
    q[j][2] = ((j+1)*10)+1;
    j++;
}
i++;
}
printf("\n\t Transition State Table \n");
printf("_______________________________\n");
printf("current state |\t Input state |\t Next state ");
printf("\n _______________________________\n");
for(i=0; i<=j ; i++){
    if(q[i][0]!=0) printf("\n q[%d]\t | a | q[%d]", i, q[i][0]);
    if(q[i][1]!=0)printf("\n q[%d]\t | b | q[%d]", i, q[i][1]);
```

```
if(q[i][2]!=0)
{
  if(q[i][2]<10) printf("\nq[%d]\t|c|q[%d]",i, q[i][2]);
  else printf("\nq[%d]\t|c|q[%d]",i, q[i][2]/10, q[i][2]%10);
}
}
printf("\n_____\n");
return 0;
}
```

Result:-

Program runs successfully. Thus RE to NFA conversion code implemented & runs successfully.



sample input: (a|b)*

sample output :

Transition Table

current state	Input	Next state
q[0]	e	q[1], q[1]
q[1]	e	q[2], q[4]
q[2]	a	q[3]
q[3]	e	q[6]
q[4]	b	q[5]
q[5]	e	q[6]
q[6]	e	q[1], q[1]

Verified

Experiment-3

Conversion of NFA to DFA.

Date:- 02/02/23

Aim:- To generate a code for the implementation to convert NFA to DFA.

Algorithm:-

(1) Initially $Q' = Q$

(2) Add q_0 of NFA to Q'. Then find the transitions from this start state

(3) In Q', find the possible set of states for each input symbol. If this set of states is not in Q', then add it to Q'.

(4) In DFA, the final state will be all the state which contain final states of NFA.

Code:-

```
#include<iostream>
#include<bits/stdc++.h>
using namespace std;
void print(vector<vector<vector<int>>>table){
    cout<<"STATE INPUT:";
    char a='a';
    for(int i=0;i<table[0].size()-1;i++)
        cout<<"  "<<a++<<" |";
    }
    for(int i=0;i<table.size();i++){
        cout<<"  "<<i<<"  ";
        for(int j=0;j<table[i].size();j++){
            cout<<" \t";
            for(int k=0;k<table[i][j].size();)
             cout<<table[i][j][k]<<"  ";
        }
    }
    cout<<endl;
}
}
void printdfa(vector<vector<int>>states vector<vector<int>>dfa)
{ cout<<"state/input:";
  char a='a';
```

```
for(int i=0; i<dfa[0]; i++) {
        cout<<"{";
for(int h=0; h<states[i].size();h++){
      cout<<states[i][h]<<" "; }
  if(states[i].empty()) {
          cout<<"A";
   }
   cout<<"}";
   for(int j=0; j<dfa[i].size(); j++) {
       cout<<"\";
       for(int k=0; k<dfa[i][j].size(); k++) {
          cout<<dfa[i][j][k]<<" ";
       }
       if(dfa[i][j].empty()){
            cout<<"A";
       }
     } cout<<endl;
    }
 }
int main(){
int n, alpha;
cout<<"Enter total no. of states in NFA: ";
cout<<"Enter the no. of elements in the alphabets: ";
vector<vector<vector<int>> table;
for(int i=0; i<n; i++){
   cout<<"For state: "<<i<<endl;
   vector<vector<int>>v;
   char a='a';
   int y, yn;
   for(int j=0; j<alpha; j++){
     vector<int> t;
     cout<<"Enter output states: "<<endl;
     for(int k=0; k<yn; k++)
        cin>>y;
        t.pushback(y);
    }
    v.pushback(t);
  }
   vector<int>t;
   table.push.back(v);
}
```

```
print(table);
vector<vector<vector<int>>> dfa;
vector<vector<int>> states;
state.push_back (closure(0,table));
queue<vector<int>> q;
q.push (state[0]);
while (!q.empty()) {
   vector<int> f = q.front();
   q.pop();
   for(int i=0; i<alpha; i++) {
        vector<int> t;
        set<int> s;
        for(int j=0; j<f.size(); j++) {
          for (set<int>::iterator u = s.begin();
          u != s.end(); u++)
            t.push_back (*u);
            v.push_back(t);
         if (find(state.begin(), states.end(), t)
         {                                  == states
             states.push_back(t);
             q.push(t); }
             printdfa(states, dfa); }
```

Result:-

Program to implement NFA to DFA conversion was implemented.

SAMPLE OUTPUT & INPUT :-

I/P :- state name: A
path : 0

Enter end state from state A travelling through path 0:
A
path : 1
Enter end state from state A travelling through path 1:
A B
state name : B
path : 0
Enter end state from state B travelling through path 0 :
C
path : 1
Enter end state from state C travelling through path 0:
path : 1
Enter end state from state C travelling through path 1:

NFA :-

{'A': {'0': ['A'], '1': ['A', 'B']}, 'B': {'0': ['C'], '1': ['C'
'C': {'0': [], '1': []}}

Printing NFA table:-

	0	1
A	[A]	[A,B]
B	[C]	[C]
C	[]	[]

Enter final state of NFA :

C

SAMPLE OUTPUT:-

DFA :-

{'A': {'0': 'A', '1': 'AB'}, 'AB': {'0': 'AC', '1': 'ABC'},
'AC': {'0': 'A', '1': 'AB'}, 'ABC': {'0': 'AC', '1': 'ABC'}}

Printing DFA table:-

	0	1
A	A	AB
AB	AC	ABC
AC	A	AB
ABC	AC	ABC

Final states of the DFA are :- ['AC', 'ABC']

Experiment 4(a) & 4(b)

Elimination of left recursion & left factoring

Aim:- 4(a) A program for elimination of left recursion.

Algorithm:-

1. Start the program
2. Initialize the arrays for taking input from the user.
3. Prompt the user to input the no. of terminals having left recursion & 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:-

   $A \rightarrow A\alpha_1 | A\alpha_2 | \ldots | A\alpha_m$

   $A \rightarrow \beta_1 | \beta_2 | \cdots | \beta_m$

   Then replace it by

   $A \rightarrow \beta_i A'$ $i = 1, 2, 3, \cdots, m$

   $A' \rightarrow \alpha_j A'$ $j = 1, 2, 3, \cdots, n$

   $A' \rightarrow \epsilon$.
6. After eliminating the left recursion by applying these rules, display the productions without left recursion.
7. Stop.

Program:-

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define SIZE 20
int main()
{
    char pro[SIZE], alpha[SIZE], beta[SIZE];
    int nont-terminal, i, j, index = 3;
    printf("Enter the production as E->E|A: ");
    scanf("%s", pro);
    nont-terminal = pro[0];
    if(nont-terminal == pro[index])
```

```
{
  for(i=++index, j=0; pro[i] != '|'; i++, j++) {
      alpha[j] = pro[i];
      if(pro[i+1] == 0) {
          printf("This Grammar CAN'T BE REDUCED.\n");
          exit(0);
      }
  }
  alpha[j] = '\0';
  if(pro[++i] != 0)
  {
      for(j=1, i=0; pro[j] != '\0'; i++, j++) {
          beta[i] = pro[j];
      }
      beta[i] = '\0';
      printf("\nGrammar without Left Recursion:\n\n");
      printf("%c -> %s%c'\n", non-terminal, beta,
                              non-terminal);
      printf("%c -> %s%c' | #\n", non-terminal, alpha,
                              non-terminal);
  }
  else
      printf("This Grammar CAN'T be REDUCED.\n");
  }
  else
      printf("\n This Grammar is not LEFT RECURSIVE.\n");
}
```

Exp: 4(b)

Aim:- A program for the implementation of left factoring

Code:-

```
#include<iostream.h>
#include<stdio.h>
int main()
{
  char gram[20], part1[20], part2[20], modifiedGram[20], newGram
  [20], tempGram[20];
  int i, j=0, k=0, l=0, pos;
  printf("Enter Production: A->");
  gets(gram);
  for(i=0; gram[i]!='|'; i++, j++)
      part1[j] = gram[i];
  part1[j]='\0';
  for(i=0; gram[j]!='\0'; j++, i++)
      part2[i] = gram[j];
  part2[i]='\0';
  for(i=0; i<strlen(part1)||i<strlen(part2); i++){
    if(part1[i]==part2[i]){
      modifiedGram[k] = part1[i];
      k++;
      pos = i+1;
    }
  }
  for(i=pos, j=0; part1[i]!='\0'; i++, j++){
      newGram[j] = part1[i];
  }
  newGram[j++]='|';
  for(i=pos; part2[i]!='\0'; i++, j++){
      newGram[j] = part2[i];
  }
```

```
modifiedGram[k] = 'x';
modifiedGram[++k] = '\0';
newGram[j] = '\0';
printf("\nGrammar without left factoring : \n");
printf("A -> %s", modifiedGram);
printf("\nX -> %s\n", newGram);
}
```

Result:-

Left factoring & left recursion was implemented successfully.

4(a). Removal of Left Recursion

Output:-

Enter the Production as E-->E|A; S->Sa|a|b.
Grammar without Left Recursion:-

S -> a|bS'
S' -> aS'

o/p verified

Dnw 13/2

4(b) Left Factoring

Enter production : A -> bE+acF|bE+f
Grammar without Left Factoring :

A -> bE+X
X -> acF|f.

o/p verified

Dnw 13/2

Output:

Enter the productions:-

S → aBDh

B → cC

C → bC

D → EF

E → g

f → f

end

S	a	#
B	c	g
C	b	g
D	g	h
E	g	f
F	f	h

Experiment-5

First and Follow.

Date:- 20/02/2023

Aim:- To write a program to perform first and follow using any language

Algorithm:-

For computing the first:-

1. If X is a terminal then first(x) = {x}
   Example: F → I|id . We can write it as first(F) ⇒ {(, id)
2. If X is a non-terminal like E → T then go to FIRST | substitute T with other productions until you get a terminal as the first symbol.
3. If X → ϵ then add ϵ to FIRST(x).

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<iostream.h>
#include<string.h>
#define max 20
using namespace std;
char prod[max][10];
char ter[10], nt[10];
char first[10], follow[10][10];
int eps[10];
int count_var=0;
```

```
int findpos(char ch){
    int n;
    for(n=0;nt[n]!='\0';n++)
        if(nt[n]==ch) break;
        if(nt[n]=='\0')return1;
        return n;
}
int Iscap(charc){
    if(c>='A'&& c<='z')
      return1;
    return 0;
}
void add(char *arr, charc){
    int i; flag=0;
    for(i=0; arr[i]!='\0';i++){
      if(arr[i]==c){
          flag=1;
          break;
      }
    }
    if(flag!=1)arr[strlen(arr)]=c;
}
void addarr(char *s1, char *s2){
    int i,j,flag=99;
    for(i=0; s2[i]!='\0';i++){
        flag=0
        for(j=0;;j++){
          if(s2[i]==s1[j]){
            flag=1;
            break;
        }
        if(j==strlen(s1)&&flag!=1){
            s1[strlen(s1)]=s2[i];
            break;
        }
    }
}
```

```
void addprod(char *s){
   int i;
   prod[count_var][0] = s[0];
   for(i=3; s[i]!='\0'; i++) {
      if(!Iscap(s[i])) add(ter, s[i]);
      prod[count_var][i-2] = s[i];
   }
   prod[count_var][i-2] = '\0';
   add(nt, s[0]);
   count_var++;
}

void findfirst(){
    int i,j,n,k,e,n1;
    for(i=0;i<count_var;i++){
      for(j=0;j<count_var;j++){
         n=findpos(prod[j][0]);
         if(prod[j][i]==(char)238) eps[n] =1;
         else {
           for(k=1,e=1;prod[j][k]!='\0'&& e==1;k++){
            if(!Iscap(prod[j][k])){
               e=0;
               add(first[n],prod[j][k]);
            }
            else{
               n1=findpos(prod[j][k]);
               addarr(first[n],first[n1]);
               if(eps[n1]==0)
                  e=0;
            }
         }
         if(e==1) eps[n]=1;
      }
   }
}
```

```
void findfollow(){
 int i, j, k, n, e, n1;
 n=findpos(prod[0][0]);
 addfollow[n], '#');
 for(i=0; i<count_var; i++){
    for(j=0; j<count_var; j++){
      k=strlen(prod[j])-1;
      for(; k>0; k--){
        if(Iscap(prod[j][k])){
          n=findpos(prod[j][k]);
          if(prod[j][k+1]=='\0')
          {
            n1=findpos(prod[j][0]);
            addarr(follow[n], follow[n1]);
          }
          if(Iscap(prod[j][k+1]))
          { n1=findpos(prod[j][k+1]);
            addarr(follow[n], first[n1]);
            if(eps[n1]==1)
            {
              n1=findpos(prod[j][0]);
              addarr(follow[n], follow[n1]);
            }
          }
          else if(prod[j][k+1]!='\0')
            add(follow[n], prod[j][k+1]);
      }
    }
 }
}
```

```
int main()
{
    char s[max], i;
    cout<<"Enter the productions \n";
    cin>>s;
    while(strcmp("ends", s)) {
        addprod(s);
        cin>>s;
    }
    findfirst();
    findfollow();
    for(i=0; i<strlen(nt); i++) {
      cout<<nt[i]<<"\t";
      cout<<first[i];
      if(eps[i]==1) cout<<(char)238)<<"\t";
      else cout<<"\t";
      cout<<follow[i]<<"\n";
    }
    return 0;
}
```

Result :- Firsts and follows sets of the non-terminal of a grammar were found successfully.

22/2/23

Output :-

Enter the productions :

S → aBDh

B → cC

C → bC

D → EF

E → g

F → f

end

S	a	#
B	c	g
C	b	g
D	g	h
E	g	f
F	f	h

Experiment-6

Predictive Parsing Table

Date:- 27/02/23

Aim:- A program for Predictive Parsing.

Algorithm:-

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 → α in G){
 for(each terminal a in FIRST(α))
  add A → α to M[A,a];
  if(ε is in FIRST(α))
  for(each symbol b in FOLLOW(A))
  add A → α to M[A,b];
```

5. Print the resulting stack.
6. Print if the grammar is accepted or not.
7. Exit the program.

Code:-

```
#include<iostream.h>
#include<string.h>
using namespace std;
int n,n1,n2;
int getPosition(string arr[], string q, int size)
{  for(int i=0;i<size;i++)
    {  if(q==arr[i])
          return i;
    }  return -1;
}
int main()
{  string prods[10], first[10], follow[10], nonterms[10], terms[10]
   string PP_table[20][20]={};
   cout<<"Enter the number of productions :";
   cin>>n;
   cin.ignore();
   cout<<"Enter the productions"<<endl;
```

```
for(int i=0; i<n; i++)
{ getline(cin,prods[i]);
  cout<<"Enter first for"<<prods[i].substr(3)<<":";
  getline(cin,first[i]);
}
cout<<"Enter the no. of terminals :";
cin>>n2;
cin.ignore();
cout<<"Enter the terminals "<<endl;
for(int i=0; i<n2; i++)
{ cin>>terms[i];
}
terms[n2]="$";
n2++;
cout<<"Enter the no. of non-terminals :";
cin>>n1;
cin.ignore();
for(int i=0; i<n1; i++)
{ cout<<"Enter Non-terminal:";
  getline(cin, nonterms[i]);
  cout<<"Enter follow of "<<nonterms[i]<<":";
  getline(cin,follow[i]);
}
cout<<endl;
cout<<"Grammar"<<endl;
for(int i=0 ; i<n; i++)
{ cout<<prods[i]<<endl;
}
for(j=0 ; j<n ; j++)
{ int row= getposition(nonterms, prods[j].substr(0,1), n1);
  if(prods[j].at(3)!="#")
  { for(int i=0; i<first[j].length(); i++)
    { int col = getposition(terms, first[j].substr(i,1), n2);
      pp.table[row][col] = prods[j];
    }
  }
  else
  { for(int i=0; i<follow[row].length(); i++)
```

```
        int col= getPosition(terms,follow[row].substr(i,1),n2);
        PP_table[row][col]=prods[j];
      }
    }
  }
  for(int j=0;j<n2;j++)
      cout<<"\t"<<terms[j];
  cout<<endl;
  for(int i=0;i<n1;i++)
  { cout<<nonterms[i]<<"\t";
    for(int j=0;j<n2;j++)
    { cout<<PP_table[i][j]<<"\t";
    }
    cout<<endl;
  }
  char c;
  do{
  string ip;
  deque<string> PP_stack;
  PP_stack.push_front("$");
  PP_stack.push_front(prods[0].substr(0,1));
  cout<<"Enter the string to be parsed :";
  getline(cin,ip);
  ip.push_back('$');
  ip.push_back('$');
  cout<<"stack\tInput\tAction"<<endl;
  while(true)
  { for(int i=0;i<PP_stack.size();i++)
      cout<<PP_stack[i];
    cout<<"\t"<<ip<<"\t";
    int row1 = getPosition(nonterms, PP_stack.front(), n1);
    int row2= getPosition(terms, PP_stack.front(), n2);
    int column = getPosition(terms, ip.substr(0,1), n2);
    if(row1!=-1&& column!=-1)
    { string p = PP_table[row1][column];
      if(p.empty())
      { cout<<endl<<"string cannot be parsed."<<endl;
        break;
      }
```

```
pp-stack.pop_front();
if(p[3] != 'H')
{ for(int x=p.size()-1; x>2; x--)
  { pp-stack.push_front(p.substr(x,1));
  }
}
cout<<p;
}
else
{ if(ip.substr(0,1)==pp-stack.front())
  { if(pp-stack.front()=="$")
    { cout<<endl<<"string Parsed"<<endl;
      break;
    } cout<<"match"<<ip[0];
      pp-stack.pop_front();
      ip=ip.substr(1);
    } else
    { cout<<endl<<"string cannot be Parsed."<<endl;
      break;
    }
  }
  cout<<endl;
} cout<<"continue?(Y/N)";
  cin>>c;
  cin.ignore();
} while(c=='Y' || c=='y');
  return 0;
}
```

Result:- Program for Predictive Parsing was implemented successfully

27/2/23

Output :-

Enter the no. of nonterminals
2
Enter the no. of productions in the grammar :-
S → CC
C → eC | d

First
FIRS[S] = ed
FIRS[C] = ed
Follow
FOLLOW[S] = $
FOLLOW[C] = ed$
M[S,e] = S → CC
M[S,d] = S → CC
M[S,e] = C → eC
M[C,d] = C → d

A → αBβ
S → CC
C → eC | d
= {e, d, $}
A → α B β
S → C C

O/P verified
27/2

	d	e	$
S	S → CC	S → CC	
C	C → d	S → eC	

Experiment-7

Date:- 06/03/23

Shift Reduce Parsing

Aim:- To implement shift reduce parsing

Algorithm:-

1. Shift reduce parsing is a process of reducing a string to the start symbol of a grammar.
2. Shift reduce parsing uses a stack to hold the grammar & an input tape to hold the string.
3. Two actions are performed: shift & reduce
4. on each shift action, the current symbol is pushed to a stack
5. At each reduction, the symbols will be replaced by non-terminals. The symbol is to the right side of the production & non-terminal to the left side of the production.

Code:-

```
#include<stdio.h>
#include<iostream>
struct strul
{
  char non_term[1], pro[25];
} cfg[25];
int n, st=-1, j, i, t=-1, m;
char str[20], stack[20], ch, tmp[20];
void match(int k);
void matchl(int k);
int main()
{
   printf("Enter the no. of productions:\n\r");
   scanf("%d",&n);
   printf("Enter the productions on left and right sides:\n");
   for(i=0; i<n; i++)
   { scanf("%s", cfg[i].non_term);
     scanf("%s", cfg[i].pro);
   }
   printf("Enter the input string:\n");
   scanf("%s", str);
```

```
i=0;
do
{   ch=str[i];
    stack[++st]=ch;
    tmp[0]=ch;
    match(1);
    i++;
}while(str[i]!='\0');
    c=st;
    y=st;
    puts(stack);
    while(y<=c)
    {  tmp[t++]=stack[y++];
       p++;
    }
 match1(p);
}
cfg[0].non_ter[i]='\0';
if(strcmp(stack, cfg[0].non_ter)==0)
   printf("string is present in Grammar\n");
 else
   printf("string is not present in Grammar\n");
 return 0;
}
void match(int k)
{ for(j=0;j<n;j++){
     if(strlen(cfg[j].pro)==k){
        if(strcmp(tmp.cfg[i].pro)==0)
        { stack[st]=cfg[j].non_ter[0];
          break;
        }}}}
```

```
void match1(int k)
{
  int x=1, y;
  y = k-1;
  for (j=0; j<n; j++)
  { if (strlen(cfg[j].pro)==k)
    { if (strcmp(tmp, cfg[j].pro)==0)
      { k=c-k+1;
        stack[k] = cfg[j].non_ter[0];
        do {
          stack[k+x] = '\0';
          tmp[t--] = '\0';
          c--;
          x++; }
          while(x<=u);
          tmp[t] = '\0';
          puts(stack);
          break;
      }
    }
  }
}
```

Result:-

code was implemented successfully

sample output

Enter the no. of productions
3
Enter the no. of productions on left & right nodes
E → E+E
E → E*E
E → i
Enter the input string
i+i*i
E+E*E
E+E
E

string is present in grammar G.

Experiment-8

Date:- 14/03/25

Leading & Trailing

Aim:- A program to implement Leading & Trailing

Algorithm :-

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 it's 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.
7. Stop.

Code:-

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
int vars,terms,i,j,k,m,rep,count,temp=-1;
char var[10],term[10],lead[10],lead[10][10],trail[10][10];
struct grammar
{ int prodno;
  char lhs,rhs[20][20];
} gram[50];
void get()
{
   cout<<"\n---LEADING & 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<<"\n\n---- PRODUCTION 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]!='\0')
        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]!='\0')
        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]<<",";
        }
    }
}
void main()
{   clrscr();
    get();
    leading();
    trailing();
    display();
    getch();
}
```

Result:-

The program was successfully complied & run.

sample output

Enter the no. of non-terminals = 3
Enter the Non-Terminals :
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 productions 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

Experiment-9

Date:- 21/03/23

Computation of LR(0) Items.

Aim:- A program to implement LR(0) items.

Algorithm:-

1. Start.
2. Create structure for production with LHS & RHS.
3. open file and read input from file.
4. Build state 0 from extra grammar law S'→S$ that is all start symbol of grammar and one Dot (.) 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 & 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.h>
#include<conio.h>
#include<string.h>
char prod[20][20], listofvar[26]="ABCDEFGHIJKLMNOPQR";
int novar=1, i=0, j=0, k=0, n=0, m=0, arr[30];
int noitem=0;
```

```
struct Grammar
{ char lhs;
  char rhs[8];
} g[20], item[20], clos[20][10];
int isvariable(char variable)
{ for(int i=0; i<novar; i++)
    if(g[i].lhs == variable)
      return i+1;
    return 0;
}
void findclosure(int z, char a)
{ int n=0, i=0, j=0, k=0, l=0;
  for(i=0; i<arr[z]; i++)
  { for(j=0; j<strlen(clos[z][i].rhs); j++)
    { if(clos[z][i].rhs[j]=='.' && clos[z][i].rhs[j+1]==
      { 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;
}
void main()
{  clrscr();
   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];
          }
      }
  }
```

Result :-

Code was implemented successfully.

21/3

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

augmented 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→.(f)
f→.i

I5

F → i.

I6

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).

Experiment-10

Date:- 29/03/23

Intermediate code generation

Aim:- A program to implement Intermediate code generation: Postfix, Prefix.

Algorithm:-

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. 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 & output it until a '(' is encountered, and discard both the paranthesis.
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 empty, then simply push the operator in the stack.
16. Repeat steps 6 to 9 until the stack is empty.

Code :-

```
#include <bits/stdc++.h>
using namespace std;
int precedence(char ch)
{
  if (ch == '^')
    return 3;
  elseif (ch == '/' || ch == '*')
    return 2;
  else if(ch == '+' || ch == '-')
    return 1;
  else
    return -1;
}
string infixToPostfix (string s)
{ stack <char> st;
  string postfix_exp;
  for(int i=0; i<s.length(); i++)
   { char ch = s[i];
    if ((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'z') ||
        (ch >= '0' && ch <= '9'))
        postfix_exp += ch;
      else if (ch == '(')
       st.push('(');
      else if (ch == ')') {
        while (st.pop() != '(')
          postfix_exp += st.top();
          st.pop ();
        }
        st.pop();
      }
```

```
else{
  while(!st.empty() && precedence(s[i]<=precedence(st.top()))
    { postfix_exp+=st.top();
      st.pop();
    }
    st.push(ch);
  }}
  while(!st.empty())
  {
    postfix_exp+=st.top();
    st.pop();
  }
    return postfix_exp;
  }
void postfixToprfix()
{ int n,i,j=0;
  char c[20];
  char a,b, op;
  cout<<"Enter postfix expression :\n";
  gets(s);
  n=strlen(str);
  for(i=0;i<MAX;i++)
    stack[i]='\0';
  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=0;
    j=strlen(c)-1;
    char d[20];
     while(c[i]!='\0'){
        d[j--] = c[i+1];
     }
     cout<<d;
}
int main()
{   string infix_expression;
    cin>> infix_expression;
    cout<< "The postfix string is : "<<infixToPostfix(infix_expression);
    return 0;
}
```

Result:-

code was implemented successfully.

Output :-

A+B\C*D

Postfix string is :- AB+CD*\

Prefix string is :- \+AB*CD.

Experiment-11. Date:-05|04|23

Intermediate Code Generation:-Quadruple, Triple, Indirect triple

Aim:- Intermediate code generation:- Quadruple, Triple, Indirect triple

Algorithm :-

The algorithm takes a sequence of three address statements as input. For each three address statements of the form a := b op c perform the various actions. These are as follows: 1. Invoke a function getreg to find out the location L where the result of the computation b 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 mov y', L to place a copy of y in L.

3. Generate the instruction 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 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 op z those register will no longer contain y or z.

Program :-

```
#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("\t the 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];
    printf(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]=='+')
}
void small()
{ pi=0; l=0;
  for(i=0; i<strlen(j); i++)
  { for(m=0; m<5; m++)
    if(j[i]==sku[m])
    if(pi<=p[m])
    {
      pi=p[m];
      l=1;
      k=i;
    }
  }
  if(l==1)
  dove(k);
  else
  exit(0);
}
```

Result:-

Program was successfully compiled & run.

Output :-

Enter the expression : a = b+c-d

The intermediate code is:-

t1 = b+c
t2 = t1-d
a = t2

Experiment-12

simple code Generator

Date:- 13/04/23

Aim:- WAP to implement simple code Generator.

Algorithm:-

1. Start
2. Get the address code sequence
3. Determine current location of 3 using address
4. If current location not already exist generate move(B, 0).
5. Update address for A(for 2nd operand).
6. If current value of B and () is null, exist.
7. If they generate operator ().A, 3 ADPR
8. Store the mov einstruction in memory
9. Stop.

Code:-

```
#include <stdio.h>
#include<stdlib.h>
#include<string.h>
#define MAX_LINES 100
#define MAX_LINE_LENGTH 50
void generateAssembly (char code[])
    char arg1[10], arg2[10], result[10], op[2];
    sscanf(code, "%s = %s %s %s", result, arg1, op, arg2);
    printf("mov AX, %s\n", arg1);
    if (strcmp (op, "+") ==0) {
        printf("ADD AX, %s\n", arg2);
    }elseif (strcmp (op, "-") ==0){
        printf("SUB AX, %s\n", arg2);
    } elseif(strcmp(op, "/") ==0) {
        printf("mov DX, 0\n")
        printf ("DIV AX %s\n", arg2);
    }
```

```
    printf("mov %s, AX \n", result);
}

int main()
{ char code[MAX_LINES][MAX_LINE_LENGTH];
  int num_lines = 0;
  printf("Enter 3 address code (result=arg1 op arg2), or
      type \"done\" to finish: \n");
  while (num_lines < MAX_LINES) {
      fgets(code[num_lines], MAX_LINE_LENGTH, stdin);
      if(strcmp(code[num_lines], "done", 4)==0) {
          break;
      }
      num_lines++;
  }
  for(int i=0; i<num_lines; i++) {
      generateAssembly(code[i]);
      printf("\n");
  }
  return 0;
}
```

Result :-

Program was implemented successfully.

Output :-

Enter 3-address code (result=arg1 op arg2) or type "d
to finish:

```
a = b * c
c = a * c
done
mov AX, b
MUL AX, c
mov a, AX
mov AX, a
MUL AX, c
mov c, AX.
```

o/p verified
13/4

Experiment - 13 Date:- 21/04/23

Implementation of DAG.

Aim:- To study and implement Directed Acyclic Graphs.

Algorithm :-

1) The leaves of a graph are labelled by a unique identifier & that identifier can be variable names or constants.
2) Interior nodes of the graph are labelled by an operator symbol.
3) If y operand is undefined then create node(y).
4) If z operand is undefined then for case(i) create node(z).
5) For case(i) create node(op) whoose right child is node(z) and left child is node(y).
6) For case (ii) check whether there is node (op) with one child node(y).
7) For case(iii), node n will be node(y).
8) For node(x), delete x from the list of identifiers. Append x to attached identifiers list for the node n found in step 2. Finally set node(x) to n.

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;
   cin>>n;
   string st[n];
   for(int i=0;i<n;i++){
      cout<<"Enter string value for st["<<i<<"]: ";
      cin>>st[i];
 }
 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;
}
```

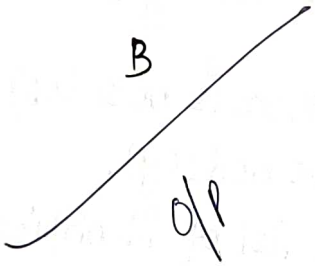
Result:-

Code was implemented & studied successfully

Output :-

Enter string value for st[0] : A = x+y
Enter string value for st[1] : B = A * z
Enter string value for st[2] : C = B | x

label	ptr	leftPtr	rightPtr
A	+	x	y
B	*	A	z
C	|	B	x.



MINI PROJECT