A Practical Report Submitted in fulfilment of the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

Year 2023-2024

By

Mr. YADAV YOGESH RAMASHREY MEENADEVI
(Application ID: 32579)

Seat no . 4500070

Under guidance of

Prof. BHARATI GAIKAR

Institute of Distance and Open Learning,
Vidya Nagar, Kalina, Santacruz East- 400098.
University of Mumbai

INSTITUTE OF DISTANCE AND OPEN LEARNING

Vidya Nagari, Kalina, Santacruz East – 400098

CERTIFICATE

This is to certify that,

Mr. **YADAV YOGESH RAMASHREY MEENADEVI**,
Application ID: **32579,**

Student of Master of Science in Computer Science has satisfactorily completed the practical in

Design and Implementation of Modern Compilers.

Name	**Application ID**
Mr. YADAV YOGESH RAMASHREY MEENADEVI	32579

Subject In-charge

Examiner

INDEX

PRACTICAL 01

AIM: Write a program to convert the given NDFA to DFA.

THEORY:

Finite Automata:

It is a mathematical model of a system with discrete input and output. It recognizes the tokens. There are two types of finite automata,

- Nondeterministic finite automata (NFA)
- Deterministic finite automata (DFA)

Nondeterministic finite automata (NFA):

A nondeterministic finite state machine or nondeterministic finite automaton (NFA) is a finite state machine where for each pair of state and input symbol there may be several possible next states. Non-deterministic finite state machines are sometimes studied by the name sub shifts of finite type.

A nondeterministic finite automata is a quintuple <Q,∑,δ,q0, F>; where:

Q is a finite set of states.

∑ is a finite set of input symbols.

δ is the, possibly partial, transition function.

q0 element of Q is called the initial state.

F contained in Q is called the set of final states.

Deterministic finite automaton (DFA):

A deterministic finite state machine or deterministic finite automaton (DFA) is a finite state machine where for each pair of state and input symbol there is one and only one transition to a next state. DFAs recognize the set of regular languages and no other languages.

A DFA is a 5-tuple, (S, Σ, T, s, A), consisting of:

- a finite set of states (Q)
- a finite set called the alphabet (Σ)
- a transition function (T : S × Q)
- a start state (s ∈ S)
- a set of accept states (A ⊆ S)

Algorithm for NFA to DFA conversion:

Input: NFA N

Output: DFA D accepting the same language

```
Computation of ε – CLOSURE:
begin
        push all states in T onto STACK;
        ε – CLOSURE(T) := T;
        while STACK not empty do
                begin
                        pop s, the top element of STACK, off of STACK;
                        for each state t with an edge from s to t labeled ε do
                                if t is not in ε – CLOSURE(T) do
                                        begin
                        add t to ε – CLOSURE(T);
                        push t onto STACK;
                    end
        end
end
```

Subset construction algorithm:

```
while there is an unmarked state x={s1,s2,…,sn} of D do
    begin
        mark x;
        for each input symbol a do
            begin
                let T be the set of states to which there is a transition on a
                from some state si in x;
                y:= ε – CLOSURE(T);
                if y has not yet been added to the set of states of D then
                    make y an "unmarked" state of D;
                    add a transition from x to y labeled a if not already
                    present;
            end
    end
```

SOURCE CODE:

```
#include<iostream.h>
#include<conio.h>
class DFA;
int closure[20],global=0;
class NFA
{
 protected:
  struct first
  {
   int no_of_o_s;
   int output_states[20];
  }tt[20][5];
   int no_states,no_inps;
   int start_state,no_final_states,final_states[10];
   char inputs[5];
   public:
    void init1(void);
    void Eclosure(int state_no);
    void printTT(void);
   friend void Conversion(NFA*N,DFA*D);
   friend int FindTransaction(NFA*N,DFA*D,int curr,int arr[20],int input);
   friend int FindEpsi(NFA*N,int arr1[15],int no,int arr2[20]);
   friend void Addstate(NFA*N,DFA*D,int curr,int arr[15],int no,int input,int found);
  };
  void NFA :: init1(void)
   {
```

```
int i,j,k;
cout<<"The NFA values \n";
cout<<"Enter no of states: ";
cin>>no_states;
cout<<"Enter start_state: ";
cin>>start_state;
cout<<"Enter no of final states: ";
cin>>no_final_states;
cout<<"Enter that final states: ";
for(i=0;i<no_final_states;i++)
 cin>>final_states[i];
 cout<<"Enter no of inputs: ";
 cin>>no_inps;
 cout<<"Enter the input symbols: \n";
for(i=0;i<no_inps;i++)
 cin>>inputs[i];
 cout<<"Enter the transition: \n";
for(i=0;i<no_states;i++)
 {
  cout<<"For state: "<<i<<"\n";
  cout<<"-----------"<<"\n";
  for(j=0;j<no_inps;j++)
   {
         cout<<"Enter no of output state for input symbol: "<<inputs[j]<<"= ";
         cin>>tt[i][j].no_of_o_s;
         for(k=0;k<tt[i][j].no_of_o_s;k++)
          {
           cout<<"Enter those states: ";
           cin>>tt[i][j].output_states[k];
          }
         }
  }
 }
 void NFA::Eclosure(int state_no)
 {
   int stack[15],top=0,pop_state,i,j,flag=0;
   global=0;
   stack[top]=state_no;
   closure[global]=state_no;
   global++;
   while(top!=-1)
   {
          pop_state=stack[top];
          top--;
          for(i=0;i<tt[pop_state][0].no_of_o_s;i++)
          {
           flag=0;
           for(j=0;j<global;j++)
             {
              if(tt[pop_state][0].output_states[i]==closure[j])
               {
                  flag=1;
                  break;
                 }
               }
               if(flag==0)
                {
                 //not in Eclosure,so add it there & then push to stack
                        top++;
                        stack[top]=tt[pop_state][0].output_states[i];
                        closure[global]=tt[pop_state][0].output_states[i];
                        global++;
                 }
              }
      }
  }
void NFA::printTT(void)
 {
  int i,j,k;
  cout<<"\n\nt\tTransition table for Given NFA\n";
  cout<<"-----------------------------------------\n";
  cout<<"states\t\t\tinputs\n";
  for(i=0;i<no_inps;i++)
   cout<<"\t\t"<<inputs[i];
   cout<<"\n-----------------------------------------\n";
   for(i=0;i<no_states;i++)
    {
     cout<<"q"<<i;
     for(j=0;j<no_inps;j++)
           {
             cout<<"\t\t";
     for(k=0;k<tt[i][j].no_of_o_s;k++)
             cout<<"q"<<tt[i][j].output_states[k]<<"";
           }
     cout<<"\n\n";
    }
    getch();
  }
  class DFA
  {
    private:
                 struct TR
                 {
                        int output_state;
                 };
                 struct Dstate
                 {
                        int no_states;
                        int states[15];
                        struct TR trn[10];
                 }Dstates[15];
                  int nDFAstates;
                  int no_final_states;
                  int final_states[10];
                  int no_inps;
                  char inputs[5];
                  public:
                          int start_state;
                          DFA(void);
                          int checkprev(int array[15],int no);
                          void PrintDFA(void);
                  friend void Conversion(NFA *N,DFA *D);
                  friend int FindTransaction(NFA *N,DFA *D,int curr,int arr[20],int input);
                  friend void Addstate(NFA *N,DFA *D,int curr,int array[15],int no,int input,int found);
                  friend int Findepsi(NFA *N,int array[15],int no,int arrayY[20]);
   };
   void main()
   {
    NFA n;
    DFA d;
    clrscr();
     n.init1();
     n.printTT();
     Conversion(&n,&d);
     d.PrintDFA();
     getch();
   }
   DFA::DFA(void)
   {
           int i,j;
           for(i=0;i<15;i++)
                        for(j=0;j<10;j++)
                                Dstates[i].trn[j].output_state=-1;
   }
   void Conversion (NFA *N,DFA *D)
   {
           int i,j,curr,found;
           int TE[20],TF[20];
           int retvalueTE,retvalueTF;
           D->start_state=0;
           D->no_final_states=0;
           D->no_inps=N->no_inps;
           for(i=1;i<N->no_inps;i++)
                        D->inputs[i]=N->inputs[i];
           global=0;
           N->Eclosure(N->start_state);//calling fn.
           curr=0;
           for(j=0;j<global;j++)
                        D->Dstates[curr].states[j]=closure[j];
           D->Dstates[curr].no_states=global;
           D->nDFAstates=curr;
           (D->nDFAstates)++;
           while(curr<D->nDFAstates)
           {
           for(i=1;i<N->no_inps;i++)
             {
                        retvalueTE=FindTransaction(N,D,curr,TE,i);
                        retvalueTF=FindEpsi(N,TE,retvalueTE,TF);
                        found=D->checkprev(TF,retvalueTF);
                        Addstate(N,D,curr,TF,retvalueTF,i,found);
                        cout<<"\ninput:"<<N->inputs[i]<<"==>TE={";
                        for(j=0;j<retvalueTE;j++)
                                cout<<TE[j]<<",";
                        cout<<"\b}";
                        cout<<"TF={";
                        for(j=0;j<retvalueTF;j++)
                                cout<<TF[j]<<",";
                        cout<<"\b}";
                        if(found==0)
                                cout<<"Not found";
                        else
                                cout<<"found";
                   }
                   curr++;
           }
   }
   int FindTransaction(NFA *N,DFA *D,int curr,int TEarray[20],int input)
   {
           int i,n,j,c;
           c=0;
           for(i=0;i<D->Dstates[curr].no_states;i++)
           {
             n=N->tt[D->Dstates[curr].states[i]][input].no_of_o_s;
             for(j=0;j<n;j++)
             {
               TEarray[c]=N->tt[D->Dstates[curr].states[i]][input].output_states[j];
               c++;
             }
           }
           return(c);
   }
   int FindEpsi(NFA*N,int TEarray[15],int nonTE,int TFarray[20])
   {
           int no_elementsTF,flag,k,i,j;
           //finding curresponding Eclosures.
                        global=0;
                        N->Eclosure(TEarray[0]);//Eclosure(8)
                        for(j=0;j<global;j++)
                                TFarray[j]=closure[j];
                        no_elementsTF=j;//no of elements in TFarray
                        for(i=1;i<nonTE;i++)
                        {
                         global=0;
                         N->Eclosure(TEarray[i]);//closure(3)={3,6,1,7,2,4}
```

```
                    for(j=0;j<global;j++)
                      {
                        flag=0;
                        for(k=0;k<no_elementsTF;k++)
                        {
                         if(closure[j]==TFarray[k])
                          {
                                flag=1;
                                break;
                          }
                        }
                        if(flag==0)
                        {
                          TFarray[no_elementsTF]=closure[j];
                          no_elementsTF++;
                        }
                      }
                   }
                         return(no_elementsTF);
}
int DFA::checkprev(int array[15],int no)
{
  int i,j,k,l;
  for(i=0;i<nDFAstates;i++)
        {
          l=0;
          for(j=0;j<Dstates[i].no_states;j++)
          {
            for(k=0;k<no;k++)
            {
              if(Dstates[i].states[j]==array[k])
              l++;
            }
          }
          if(Dstates[i].no_states==no && l==0)
          return(i);
        }
          return(0);
}
void Addstate(NFA *N,DFA *D,int curr,int array[15],int no,int input, int found)
{
  int i,j,flag;
  if(found==0)
  {
         flag=0;
         for(i=0;i<no;i++)
         {
           D->Dstates[D->nDFAstates].states[i]=array[i];
           for(j=0;j<N->no_final_states;j++)
           {
             if(D->Dstates[D->nDFAstates].states[i]==N->final_states[j])
```

```
      {
       flag=1;
       break;
      }
     }
    }
    if(flag==1)
    {
          D->final_states[D->no_final_states]=D->nDFAstates;
          D->no_final_states++;
    }
    D->Dstates[D->nDFAstates].no_states=no;
    D->Dstates[curr].trn[input].output_state=D->nDFAstates;
    D->nDFAstates++;
 }
 else
          D->Dstates[curr].trn[input].output_state=found;
}
void DFA::PrintDFA(void)
{
    int i,j;
    cout<<"\n\n\t\tTransition table for Given DFA\n";
    cout<<"----------------------------------------\n";
    cout<<"states\t\t\tinputs\n";
    for(i=1;i<no_inps;i++)
          cout<<"\t\t"<<inputs[i];
    cout<<"\n--------------------------------------\n";
    for(i=0;i<nDFAstates;i++)
    {
     cout<<"q"<<i;
     for(j=1;j<no_inps;j++)
     {
          cout<<"\t\t";
      cout<<"Q"<<Dstates[i].trn[j].output_state<<"";
     }
     cout<<"\n\n";
    }
    cout<<"----------------------------------------\n";
    cout<<"start state is:q"<<start_state<<"\n";
    cout<<"final states:{";
    for(i=0;i<no_final_states;i++)
          cout<<"q"<<final_states[i]<<",";
    cout<<"\b}\n";
    getch();
}
```

OUTPUT:

```
Inputs:
The NFA values
Enter no of states: 11
Enter start_state: 0
Enter no of final states: 1
Enter that final states: 10
Enter no of inputs: 3
Enter the input symbols:
e
a
b
Enter the transition:
For state: 0
-----------
Enter no of output state for input symbol: e= 2
Enter those states: 1
Enter those states: 7
Enter no of output state for input symbol: a= 0
Enter no of output state for input symbol: b= 0
For state: 1
-----------
Enter no of output state for input symbol: e= 2
Enter those states: 2
Enter those states: 4
Enter no of output state for input symbol: a= 0
Enter no of output state for input symbol: b= 0
For state: 2
-----------
Enter no of output state for input symbol: e= 0
Enter no of output state for input symbol: a= 1
Enter those states: 3
Enter no of output state for input symbol: b= 0
For state: 3
-----------
Enter no of output state for input symbol: e= 1
Enter those states: 6
Enter no of output state for input symbol: a= 0
Enter no of output state for input symbol: b= 0
For state: 4
-----------
Enter no of output state for input symbol: e= 0
Enter no of output state for input symbol: a= 0
Enter no of output state for input symbol: b= 1
Enter those states: 5
Enter those states: 3
Enter no of output state for input symbol: b= 0
For state: 5
-----------
Enter no of output state for input symbol: e= 1
Enter those states: 6
Enter no of output state for input symbol: a= 0
Enter no of output state for input symbol: b= 0
For state: 6
-----------
Enter no of output state for input symbol: e= 2
Enter those states: 1
Enter those states: 7
Enter no of output state for input symbol: a= 0
Enter no of output state for input symbol: b= 0
For state: 7
-----------
Enter no of output state for input symbol: e= 0
Enter no of output state for input symbol: a= 1
Enter those states: 8
Enter no of output state for input symbol: b= 0
For state: 8
-----------
Enter no of output state for input symbol: e= 0
Enter no of output state for input symbol: a= 0
Enter no of output state for input symbol: b= 1
Enter those states: 9
For state: 9
-----------
Enter no of output state for input symbol: e= 0
Enter no of output state for input symbol: a= 0
Enter no of output state for input symbol: b= 1
Enter those states: 10
For state: 10
-----------
Enter no of output state for input symbol: e= 0
Enter no of output state for input symbol: a= 0
Enter no of output state for input symbol: b= 0
```

Outputs:

```
t       Transition table for Given NFA
----------------------------------------
states                inputs
                e               a               b
----------------------------------------
q0              q1q7

q1              q2q4

q2                              q3
```

```
q3       q6

q4                      q5

q5       q6

q6       q1q7

q7               q8

q8                      q9

q9                      q10

q10

input:a==>TE={8,3}TF={8,3,6,1,7,2,4}Not found
input:b==>TE={9,5}TF={9,5,6,1,7,2,4}Not found
input:a==>TE={8,3}TF={8,3,6,1,7,2,4}Not found
input:b==>TE={5}TF={5,6,1,7,2,4}Not found
input:a==>TE={8,3}TF={8,3,6,1,7,2,4}Not found
input:b==>TE={9,5}TF={9,5,6,1,7,2,4}Not found
input:a==>TE={8,3}TF={8,3,6,1,7,2,4}Not found
input:b==>TE={10,5}TF={10,5,6,1,7,2,4}Not found
input:a==>TE={8,3}TF={8,3,6,1,7,2,4}Not found
input:b==>TE={9,5}TF={9,5,6,1,7,2,4}Not found
input:a==>TE={8,3}TF={8,3,6,1,7,2,4}Not found
input:b==>TE={5}TF={5,6,1,7,2,4}Not found
input:a==>TE={8,3}TF={8,3,6,1,7,2,4}Not found
input:b==>TE={9,5}TF={9,5,6,1,7,2,4}Not found
input:a==>TE={8,3}TF={8,3,6,1,7,2,4}Not found
input:b==>TE={10,5}TF={10,5,6,1,7,2,4}Not found
input:a==>TE={8,3}TF={8,3,6,1,7,2,4}Not found
input:b==>TE={9,5}TF={9,5,6,1,7,2,4}Not found
input:a==>TE={8,3}TF={8,3,6,1,7,2,4}Not found
input:b==>TE={5}TF={5,6,1,7,2,4}Not found
input:a==>TE={8,3}TF={8,3,6,1,7,2,4}Not found
input:b==>TE={9,5}TF={9,5,6,1,7,2,4}Not found

        Transition table for Given DFA
----------------------------------------
states        inputs
        a        b
--------------------------------------
q0       Q1       Q2

q1       Q3       Q4

q2       Q5       Q6

q3       Q7       Q8
```

```
q4          Q9          Q10

q5          Q11          Q12

q6          Q13          Q14

q7          Q7          Q8

q8          Q9          Q10

----------------------------------------
start state is:q0
final states:{q3,q6,q1,q7,q2,q4,q256,q2724}
```

PRACTICAL 02

AIM: Write a program to convert the given Right Linear Grammar to Left Linear Grammar form.

THEORY:

In a left-linear grammar, all productions have one of the two forms:

$$V \rightarrow VT^*$$

or

$$V \rightarrow T^*$$

That is, the left-hand side must consist of a single variable, and the right-hand side consists of an optional single variable followed by any number of terminals. This is just like a right-linear grammar except that, following the arrow, a variable can occur only on the left of the terminals, rather than only on the right.

PSEUDO CODE:

```
1. Represent the given left linear grammar by a transition diagram with vertices labeled by the
   non-terminal symbols and transitions labeled by the terminal symbols.
2. Interchange the position of the Initial and final state.
3. Reverse the direction of all the transitions keeping the positions of all intermediate states
   unchanged.
4. Rewrite the grammar from this Transition diagram in right linear fashion.
```

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
#define isupper(ch) (ch>=65 && ch<=90)
void main()
{
        int i,j,k,prod,flag;        char LHS[10][5],RHS[10][5],temp;
        clrscr();
        printf("\t\t\t ::: Right 2 Left Linear Conversion :::\n\n");
        printf("Enter number of Productions :");
        scanf("%d",&prod);
        printf("\aEnter Production separated by Space eg: (S aB)\n");
        for(i=0;i<prod;i++)     /*Accepting the Productions*/
        {
                printf("Enter %d Production : ",i+1);
                scanf("%s %s",&LHS[i],&RHS[i][0]);
        }
        printf("\n\aEntered Right Linear Grammer ...");
        for(i=0;i<prod;i++)     /*Printing Productions*/
        {
                printf("\n%s -> %s",LHS[i],RHS[i]);
        }
```

```
for(i=0;i<prod;i++)        /*Adding Final state Symb on RHS if NT is
                                        Absent*/
{
          flag=0,j=0;
          while(RHS[i][j]!='\0')
          {
                    if(isupper(RHS[i][j]))     /*Checking for NT*/
                    {
                              flag=1;
                              break;
                    }
                    j++;
          }
          if(flag==0)
          {
                    RHS[i][j]='Z';
                    j++;
                    RHS[i][j]='\0';
          }
}
for(i=0;i<prod;i++)        /*Reversing the RHS String*/
{
          int a=0,k=0;
          while(RHS[i][k]!='\0')
          {
                    a++;      k++;
          }
          k=0;
          while(k<a)
          {
                    temp=RHS[i][a-1];
                    RHS[i][a-1]=RHS[i][k];
                    RHS[i][k]=temp;
                    k++;      a--;
          }
}
for(i=0;i<prod;i++)         /*Interchanging LHS(NT) with RHS(NT)*/
{
          temp=LHS[i][0];
          LHS[i][0]=RHS[i][0];
          RHS[i][0]=temp;
}
for(i=0;i<prod;i++)         /*Removing Final Symb from LHS & RHS*/
{
          if(LHS[i][0]=='S')
          {
                    LHS[i][0]='Z';
          }
          else if(LHS[i][0]=='Z')
          {
                    LHS[i][0]='S';
```

```
                }
                if(RHS[i][0]=='Z')
                {
                        RHS[i][0]='S';
                }
                else if(RHS[i][0]=='S')
                {
                        RHS[i][0]='Z';
                }
        }
        for(i=0;i<prod;i++)     /*Removing Final Symb*/
        {
                j=0;
                if(RHS[i][j]=='Z')
                {
                        while(RHS[i][j]!='\0')
                        {
                                RHS[i][j]=RHS[i][j+1]; j++;
                        }
                }
        }
        printf("\n\a\nConverted Left Linear Grammer ...\n");
        for(i=0;i<prod;i++)
        {
                printf("%s -> %s\n",LHS[i],RHS[i]);
        }
        getch(); }
```

OUTPUT:

```
::: Right 2 Left Linear Conversion :::

Enter number of Productions :3
Enter Production separated by Space eg: (S aB)
Enter 1 Production : A nM
Enter 2 Production : Z jA
Enter 3 Production : M n

Entered Right Linear Grammer ...
A -> nM
Z -> jA
M -> n

Converted Left Linear Grammer ...
M -> An
A -> Sj
S -> Mn
```

PRACTICAL 03

AIM: Write a program to illustrate the generation on SPM for the input grammar.

THEORY:

A grammar is said to ge simple precedence grammar if it has no e-productions, no two productions have the same right side, and the relations $<._s$, $=_s$,and $.>_s$ and disjoint.
A SMP behaves exactly as an OPM does, but NT's are kept on the stack and enter into relations.

ALGORITHM:

Step 1: Start
Step 2: Initialize the variables.
Step 3: Left value and right value ie. lval and rval resp. of the production are separated out.
Step 4: Initialize the matrix by zero.
Step 5: For equal matrix, rval is check, their indexes are searched & 1 is returned in that particular cell.
Step 6: First of lval and rval is found out and index is set to 1.
Step 7: Call Warshall's Algorithm for First+ matrix and for First* matrix diagonally 1 is appended in the cell.
Step 8: For Last matrix, first value of lval is seen and last value of rval is seen. index of rval and lval is searched and 1 is set and transpose is taken.
Step 9: For Less matrix, multiplication of equal matrix & First+ matrix is taken.
Step10: In Greater matrix, multiplication of transpose & equal matrix is taken. and multiplication of m3 & First* is also done. Variable g is used for the storage of greater matrix and m3 is stored in g.
Step11: In superimpose matrix, assign 1 if matrix is equal, assign 2 if matrix is less and assign 3 if matrix is greater. Then display the superimpose matrix.
Step12: For parsing, initially set counter = 0. Initialize p for parsing string and hand for handle. Set front handle (fh) and back handle (bh) as –1. declare exclusive symbol as C and equal as e. S is for non-terminal and terminal. Handle is between the values < and >. Handle is compared with rval of left production and if it is found then that value of rval = lval of non-terminal. Non-terminal is replaced in parsing string instead of handle. And if equal to exclusive symbol then string is parsable else not.
Step13: Display the final matrix.
Step14: Stop.

SOURCE CODE:

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

```
#include<graphics.h>
#include<stdlib.h>
#include<alloc.h>
#include<string.h>

class spm
{
    private:
              char s[20], cnt[10], ct[10], **p1, **prod, **lval, **rval;
              char fspm[10][10];
            int meq[10][10], mfplus[10][10], mfstar[10][10];
            int ml[10][10], mltr[10][10];
              int ls[10][10], g[10][10];
            int c, i, cp, t, j, nt, n, k, l;
            int rflag, rflag1;
            int m3[10][10];
    public:
            void get_data();
            void display(int m[10][10]);
            void matrix();
            void equal();
            void first();
            void last();
            void less();
            void greater();
            void multiply(int m1[10][10], int m2[10][10]);
            void superimpose();
            void parse();
};

void spm :: get_data()
{
      cout<<"\nHow many NonTerminals:--> ";
      cin>>nt;
      cout<<"\nEnter NonTerminals: --> ";
      for(i=0; i<nt; i++)
      cin>>cnt[i];
      cout<<"\nHow many Terminals: --> ";
      cin>>t;
      cout<<"\nEnter Terminals: ";
      for(i=0; i<t; i++)
      cin>>ct[i];
      cout<<"\nHow many Productions: --> ";
      cin>>cp;
      int n1=0;
      lval=(char **)malloc(sizeof(char)*10);
      rval=(char **)malloc(sizeof(char)*10);
      for(n1=0; n1<cp; n1++)
      {
           lval[n1]=(char *)malloc(sizeof(char)*10);
           rval[n1]=(char *)malloc(sizeof(char)*10);
    }
    prod[n1]=(char *)malloc(sizeof(char)*10);
    for(n1=0; n1<cp; n1++)
    prod[n1]=(char *)malloc(sizeof(char)*10);
    for(k=0; k<cp; k++)
    {
        cout<<"\nEnter the Production: -->";
        gets(prod[k]);
        p1[k]=strtok(prod[k]," ");
        lval[k]=strtok(p1[k],"=");
        rval[k]=strtok(NULL," ");
    }
    for(k=0; k<nt; k++)
    s[k]=cnt[k];
    l=k;
    k=0;
    while(ct[k]!='\0')
    {
        s[l]=ct[k];
        k++;
        l++;
    }
}

void spm :: matrix()
{
    for(i=0; i<t+nt; i++)
    {
        for(j=0; j<t+nt; j++)
        {
            meq[i][j]=0;
            mfplus[i][j]=0;
            mfstar[i][j]=0;
            mltr[i][j]=0;
            ml[i][j]=0;
            ls[i][j]=0;
            g[i][j]=0;
            m3[i][j]=0;
            fspm[i][j]='0';
        }
    }
}

void spm :: equal()
{
    char s1, s2;
    int x=0,y=0,a=0,b=0;
    clrscr();
    for(k=0; k<cp; k++)
    {
        if(strlen(rval[k])>1)
```

```
                {
                    for(l=0; l<strlen(rval[k]); l++)
                    {
                        s1=rval[k][l];
                        if(rval[k][++l]!=NULL)
                            s2=rval[k][l];
                        else
                            break;
                        x=y=0;
                        while(s[x]!='\0')
                        {
                            if(s[x]==s1)
                            {
                                a=x;
                                break;
                            }
                            x++;
                        } //while
                        while(s[y]!='\0')
                        {
                            if(s[y]==s2)
                            {
                                b=y;
                                break;
                            }
                            y++;
                        }  //while
                       meq[a][b]=1;
                       l--;
                    } //for
                }//if
            }//for
            cout<<"Equal :\n";
           display(meq);
        }

        void spm :: first()
        {
            char f1,f2;
            int x=0,y=0,a=0,b=0;
            n=t+nt;
            for(k=0; k<cp; k++)
            {
                f1=lval[k][0];
                f2=rval[k][0];
                x=y=0;
                while(s[x]!='\0')
                {
                    if(s[x]==f1)
                    {
                        a=x;
                        break;
                    } //if
                    x++;
                } //while
                while(s[y]!='\0')
                {
                    if(s[y]==f2)
                    {
                        b=y;
                        break;
                    }
                    y++;
                }
                mfplus[a][b]=1;
            }
            int i2=0;
           while(i2<n)
           {
                for(int j2=0; j2<n; j2++)
                {
                    if(mfplus[j2][i2]==1)
                    {
                        for(int k2=0; k2<n; k2++)
                        mfplus[j2][k2]=(mfplus[j2][k2] || mfplus[i2][k2]);
                    } //if
                } //for
                i2++;
            } // while
            for(i=0; i<n; i++)
            {
                for(j=0; j<n; j++)
                mfstar[i][j]=mfplus[i][j];
                mfstar[i][i]=1;
            }
            cout<<"First+ :\n";
            display(mfplus);
            getch();
            clrscr();
            cout<<"First* :\n";
            display(mfstar);
        }

        void spm :: last()
        {
            char l1,l2;
            int x=0,y=0,a=0,b=0;
            int z;
            int n=t+nt;
            for(k=0; k<cp; k++)
            {
                l1=lval[k][0];
                z=strlen(rval[k]);
                l2=rval[k][z-1];
                x=y=0;
                while(s[x]!='\0')
                {
                    if(s[x]==l1)
                    {
                        a=x;
                        break;
                    } //if
                    x++;
                } //while
                while(s[y]!='\0')
                {
                    if(s[y]==l2)
                    {
                        b=y;
                        break;
                    }
                    y++;
                } //while
                mltr[a][b]=1;
            } //for
            int i1=0;
            while(i1<n)
            {
                for(int j1=0; j1<n; j1++)
                {
                    if(mltr[j1][i1]==1)
                    {
                        for(int k1=0; k1<n; k1++)
                        mltr[j1][k1]=(mltr[j1][k1] || mltr[i1][k1]);
                    }
                }
                i1++;
            } //while
            for(i=0; i<n; i++)
            {
                for(j=0; j<n; j++)
                ml[j][i]=mltr[i][j];
            }
            cout<<"Last T :\n";
            display(ml);
        }

        void spm :: less()
        {
            int n=t+nt;
            multiply(meq,mfplus);
            for(i=0;i<n;i++)
            {
```

```
                for(j=0;j<n;j++)
                ls[i][j]=m3[i][j];
        }
        cout<<"Less < :";
       display(ls);
}

void spm :: greater()
{

        int n=t+nt;
        flushall();
        for(i=0; i<n; i++)
        for(j=0; j<n; j++)
        m3[i][j]=0;
        multiply(ml,meq);
        multiply(m3,mfstar);
        for(i=0;i<n;i++)
        {
                for(j=0; j<n; j++)
                g[i][j]=m3[i][j];
        }
        cout<<"\n Greater > :";
        display(g);
}

void spm :: multiply(int m1[10][10],int m2[10][10])
{
        int n=t+nt;
        for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                {
                        for(int k=0;k<n;k++)
                        m3[i][j]=m3[i][j] + m1[i][k] * m2[k][j];
                        if(m3[i][j]>=2)
                        m3[i][j]=1;
                }
        }
}

void spm :: superimpose()
{
                int n=t+nt;
                for(i=0; i<n; i++)
                {
                        for(j=0; j<n; j++)
                        mltr[i][j]=0;
                }
                for(i=0;i<n;i++)
                {
```

```
            for(j=0;j<n;j++)
            {
                if(meq[i][j]==1)
                {
                    mltr[i][j]=1;
                    fspm[i][j]='=';
                }
                if(ls[i][j]==1)
                {
                    mltr[i][j]=2;
                    fspm[i][j]=(char)(238);
                }
                if(g[i][j]==1)
                {
                    mltr[i][j]=3;
                    fspm[i][j]=(char)(62);
                }
        } //for
    } //for
    display(mltr);
    cout<<"\n";
    for(i=0; i<n; i++)
    cout<<"\t"<<s[i];
    cout<<"\n  -----------------------------------------------------";
    for(i=0; i<n; i++)
    {
        cout<<"\n   "<<s[i];
        cout<<" |";
        for(j=0; j<n; j++)
        cout<<"\t"<<fspm[i][j];
    }
}
void spm :: parse()
{
    char c;
    int q=0, m=0, k=0, x=0, y=0;
    char p[15],hand[10];
    int a, b, fh=-1, bh=-1, inh=0, e=1;
    cout<<"\nEnter the string to be parsed: --> ";
    gets(p);
    cout<<"\nEnter the Exclusive symbol: --> ";
    cin>>c;
    q=strlen(p);
    for(m=0; m<q-1; m++)
    {
        x=y=0;
        while(s[x]!='\0')
        {
            if(s[x]==p[m])
            {
                a=x;
                break;
            }
            x++;
        } //while
        while(s[y]!='\0')
        {
            if(s[y]==p[m+1])
            {
                b=y;
                break;
            }
            y++;
        }
        switch(mltr[a][b])
        {
            case 2: fh=m+1;break;
            case 3: bh=m;break;
            case 1: e++;
        }
        if(fh>=0 && bh>0)
        {
            inh=fh;
            for(k=0; fh<=bh; k++, fh++)
            hand[k]=p[fh];
            hand[k]='\0';
            cout<<"\nHandle :"<<hand;
            for(k=0;k<cp;k++)
            {
                if(strcmp(rval[k],hand)==0)
                break;
            }
            p[inh]=lval[k][0];
            for(; p[bh]!='\0';)
            p[++inh]=p[++bh];
            p[bh]='\0';
            cout<<"\nP :"<<p;
            fh=bh=e=m=-1;
        } //if
    } //for
    for(k=0;k<cp;k++)
    if(strcmp(rval[k],p)==0)
    break;
    if(c==lval[k][0])
        cout<<"\n String is Parsable : --> " << lval[k][0];
    else
        cout<<"\n String is not Parsable ";
}

void spm :: display(int m[10][10])
{
    cout<<"\n\n";
```

```
        flushall();
        int p=t+nt;
        for(i=0;i<p;i++)
        cout<<"\t"<<s[i];
        cout<<"\n  ----------------------------------------";
        for(i=0; i<p; i++)
        {
            cout<<"\n   "<<s[i];
            cout<<" |";
            for(j=0; j<p; j++)
            cout<<"\t"<<m[i][j];
        }
    }
    void main()
    {
        spm s1;
        int a;
        clrscr();
        flushall();
        s1.get_data();
        s1.matrix();
        s1.equal();
        getch();
        clrscr();
        s1.first();
        getch();
        clrscr();
        s1.last();
        getch();
        clrscr();
        s1.less();
        getch();
        clrscr();
        s1.greater();
        getch();
        clrscr();
        s1.superimpose();
        getch();
        clrscr();
        s1.parse();
        getch();
   }
```

OUTPUT:

```
How many NonTerminals: --> 3

Enter NonTerminals: --> Z M L

How many Terminals: --> 4
```

```
Enter Terminals: --> a b ( )

How many Productions: --> 4

Enter the Production: --> Z=bMb

Enter the Production: --> M=a

Enter the Production: --> M=(L

Enter the Production: --> L=Ma)


Equal :

      Z    M    L    a    b    (    )
    ----------------------------------
  Z | 0    0    0    0    0    0    0
  M | 0    0    0    1    1    0    0
  L | 0    0    0    0    0    0    0
  a | 0    0    0    0    0    0    1
  b | 0    1    0    0    0    0    0
  ( | 0    0    1    0    0    0    0
  ) | 0    0    0    0    0    0    0


First+ :

      Z    M    L    a    b    (    )
    ----------------------------------
  Z | 0    0    0    0    1    0    0
  M | 0    0    0    1    0    1    0
  L | 0    1    0    1    0    1    0
  a | 0    0    0    0    0    0    0
  b | 0    0    0    0    0    0    0
  ( | 0    0    0    0    0    0    0
  ) | 0    0    0    0    0    0    0

First* :
      Z    M    L    a    b    (    )
    ----------------------------------
  Z | 1    0    0    0    1    0    0
  M | 0    1    0    1    0    1    0
  L | 0    1    1    1    0    1    0
  a | 0    0    0    1    0    0    0
  b | 0    0    0    0    1    0    0
  ( | 0    0    0    0    0    1    0
  ) | 0    0    0    0    0    0    1

Last T :
```

	Z	M	L	a	b	(	)
Z	0	0	0	0	0	0	0
M	0	0	0	0	0	0	0
L	0	1	0	0	0	0	0
a	0	1	0	0	0	0	0
b	1	0	0	0	0	0	0
(	0	0	0	0	0	0	0
)	0	1	1	0	0	0	0

Less < :

	Z	M	L	a	b	(	)
Z	0	0	0	0	0	0	0
M	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0
b	0	0	0	1	0	1	0
(	0	1	0	1	0	1	0
)	0	0	0	0	0	0	0

Greater > :

	Z	M	L	a	b	(	)
Z	0	0	0	0	0	0	0
M	0	0	0	0	0	0	0
L	0	0	0	1	1	0	0
a	0	0	0	1	1	0	0
b	0	0	0	0	0	0	0
(	0	0	0	0	0	0	0
)	0	0	0	1	1	0	0

	Z	M	L	a	b	(	)
Z	0	0	0	0	0	0	0
M	0	0	0	1	1	0	0
L	0	0	0	3	3	0	0
a	0	0	0	3	3	0	1
b	0	1	0	2	0	2	0
(	0	2	1	2	0	2	0
)	0	0	0	3	3	0	0

	Z	M	L	a	b	(	)
Z	0	0	0	0	0	0	0
M	0	0	0	=	=	0	0
L	0	0	0	>	>	0	0
a	0	0	0	>	>	0	=
b	0	=	0	e	0	e	0

```
 ( |  0    e    =    e    0    e    0
  ) | 0    0    0    >    >    0    0


Enter the string to be parsed: --> b(aa)b

  Enter the Exclusive symbol : --> Z

Handle :a
P :b(Ma)b
Handle :Ma)
P :b(Lb
Handle :(L
P :bMb
String is Parsable--> Z
```

PRACTICAL 04

AIM: Write a program to illustrate the generation on OPM for the input operator grammar.

THEORY:

DEFINITION:

Operator Precedence Grammar: -

An operator precedence grammar is a ∊-free operator grammar in which the precedence relations <, =, & > constructed. That is, for any pair of terminals a & b, never more than one of the relations
a < b, a = b & a > b is true.

Operator-Precedence Parsing:

In Operator-Precedence Parsing, we use three disjoint 'precedence relations' guide the selection of handles. If a<b, we say "a yields precedence to'b"; if a=b, 'a' has the same precedence as'b'if a>b,'a' takes precedence over 'b'.

Operator-Precedence Relations from Associativity & Precedence:

Following are some rules to select 'proper' handles to reflect a given set of associativity & precedence rules binary operators.

If operator θ1 has higher precedence than θ2, make θ1>θ2 & θ2<θ1.e.g, if * has higher precedence than +, make '* > +' & '+ < *'.

If θ1 & θ2 are operators of equal precedence, then make θ1>θ2 & θ2>θ1 if the operators are left-associative, or make θ1<θ2 & θ2<θ1 if they are right-associative.

Make θ<id, id>θ, θ<(, (<θ ,)>θ ,θ>$ & $<θ for all operators θ.

ALGORITHM:

Operator precedence matrix algorithm

```
Step 1:  Start
Step 2:  Initialize the variables.
Step 3:  Left value and right value ie. lval and rval resp. of the production are
         separated out.
Step 4:  Initialize the matrix by zero.
Step 5:  For equal matrix, rval is check, their indexes are searched & 1 is returned
         in that particular cell.
Step 6:  First of lval and rval is found out and index is set to 1.
Step 7:  Call Warshall's Algorithm for First+ matrix and for First* matrix
         diagonally 1 is appended in the cell.
Step 8:  For Last matrix, first value of lval is seen and last value of rval is seen.
         index of  rval and lval is searched and 1 is set and transpose is taken.
Step 9:  For Less matrix, multiplication of equal matrix & First+ matrix is taken.
Step10:  In Greater matrix, multiplication of transpose & equal matrix is taken.
        and multiplication of m3 & First* is also done. Variable g is used for the
        storage of greater matrix and m3 is stored in g.
Step11:  In superimpose matrix, assign 1 if matrix is equal, assign 2 if matrix is
         less and assign 3 if matrix is greater. Then display the superimpose
```

```
            matrix.
Step13:  Display the final matrix.
Step14:  Stop.
```

SOURCE CODE:

```
#include<iostream.h>
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<stdlib.h>
#include<alloc.h>
#include<string.h>
class opm
{
        private:
                        char s[20],cnt[10],ct[10],**p1,**prod,**lval,**rval;
                        int meq[10][10],mfplus[10][10],mfstar[10][10],ml[10][10];
                        int c,i,cp,t,j,nt,n,k,l;
                        int m3[10][10], m[10][10], mfterm[10][10], mlterm[10][10],
                                                                mltr[10][10];
                        int eq[10][10];
                        char sim[10][10];
        public:
                        void get_data();
                        void display(int m[10][10]);
                        void equal();
                        void first();
                        void last();
                        void multiply(int m1[10][10],int m2[10][10]);
                        void superimpose();
                        void parse();
                        void fterm();
                        void lterm();
                        void less();
                        void pequal();
                        void greater();
                        void matrix();
                        void simpose();
};
void opm::get_data()
{
        cout<<"\nHow many nonterminals:";
        cin>>nt;
        cout<<"\nEnter nonterminals:";
        for(i=0;i<nt;i++)
                cin>>cnt[i];
        cout<<"\nHow many terminals:";
        cin>>t;
        cout<<"\nEnter terminals:";
        for(i=0;i<t;i++)
```

```
                cin>>ct[i];
        ct[i]='#';
        t=t+1;
        cout<<"\nHow many productions:";
        cin>>cp;
        int n1=0;
        lval=(char **)malloc(sizeof(char)*10);
        rval=(char **)malloc(sizeof(char)*10);
        for(n1=0;n1<cp;n1++)
        {
                lval[n1]=(char *)malloc(sizeof(char)*10);
                rval[n1]=(char *)malloc(sizeof(char)*10);
        }
        prod[n1]=(char *)malloc(sizeof(char)*10);
        for(n1=0;n1<cp;n1++)
          prod[n1]=(char *)malloc(sizeof(char)*10);
        for(k=0;k<cp;k++)
        {
                cout<<"\nEnter the production:";
                gets(prod[k]);
                p1[k]=strtok(prod[k]," ");
                lval[k]=strtok(p1[k],"=");
                rval[k]=strtok(NULL," ");
        }
        for(k=0;k<nt;k++)
                s[k]=cnt[k];
        l=k;
        k=0;
        while(ct[k]!='\0')
        {
                s[l]=ct[k];
                k++;
                l++;
                cout<<"\t"<<s[l];
        }
}
void opm::matrix()
{
        for(i=0;i<t+nt;i++)
        {
                for(j=0;j<t+nt;j++)
                {
                        meq[i][j]=0;
                        mfplus[i][j]=0;
                        mfstar[i][j]=0;
                        ml[i][j]=0;
                        mfterm[i][j]=0;
                        mlterm[i][j]=0;
                        mltr[i][j]=0;
                        m3[i][j]=0;
```

```
            eq[i][j]=0;
            sim[i][j]='0';
        }
    }
}

void opm::equal()
{
    char s1,s2;
    int x=0,y=0,a=0,b=0;
    clrscr();
    for(k=0;k<cp;k++)
    {
        if(strlen(rval[k])>1)
        {
            for(l=0;l<strlen(rval[k]);l++)
            {
                s1=rval[k][l];
                if(rval[k][++l]!=NULL)
                     s2=rval[k][l];
                else
                    break;
                x=y=0;
                while(s[x]!='\0')
                {
                    if(s[x]==s1)
                    {
                        a=x;
                        break;
                    }
                    x++;
                }//while
                while(s[y]!='\0')
                {
                    if(s[y]==s2)
                    {
                        b=y;
                        break;
                    }
                    y++;
                }//while
                meq[a][b]=1;
                l--;
            }//for
        }//if
    }//for
    cout<<"Equal :\n";
    display(meq);
}
void opm::first()
```

```
{
        char f1,f2;
        int x=0,y=0,a=0,b=0;
        n=t+nt;
        for(k=0;k<cp;k++)
        {
                f1=lval[k][0];
                f2=rval[k][0];
                x=y=0;
                while(s[x]!='\0')
                {
                        if(s[x]==f1)
                        {
                                a=x;
                                break;
                        }//if
                        x++;
                }//while
                while(s[y]!='\0')
                {
                        if(s[y]==f2)
                        {
                                b=y;
                                break;
                        }
                        y++;
                 }
                 mfplus[a][b]=1;
        }
        int i1=0;
        while(i1<n)
        {
                for(int j2=0;j2<n;j2++)
                {
                        if(mfplus[j2][i1]==1)
                        {
                                for(int k2=0;k2<n;k2++)
                                        mfplus[j2][k2]=(mfplus[j2][k2] || mfplus[i1][k2]);
                        }//if
                }
                i1++;
        }
        for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                        mfstar[i][j]=mfplus[i][j];
                mfstar[i][i]=1;
        }
        cout<<"First* :\n";
        display(mfstar);
```

```
}
void opm::fterm()
{
        flushall();
        char s1,s2;
        int flag=0;
        int x=0,y=0,a=0,b=0,k1=0;
        for(k1=0;k1<cp;k1++)
        {
                flag=0;
                if(strlen(rval[k1])==1)
                {
                        for(i=0;i<nt;i++)
                        {
                                if(rval[k1][0]==cnt[i])
                                {
                                        flag=1;
                                        break;
                                }
                        }
                }
                if(flag!=1)
                {
                        s1=lval[k1][0];
                        for(i=0;i<t;i++)
                        {
                                if(rval[k1][0]==ct[i])
                                        s2=rval[k1][0];
                                else if(rval[k1][1]==ct[i])
                                        s2=rval[k1][1];
                        }
                        x=y=0;
                        while(s[x]!='\0')
                        {
                                if(s[x]==s1)
                                {
                                        a=x;
                                        break;
                                }//if
                                x++;
                        }//while
                        while(s[y]!='\0')
                        {
                                if(s[y]==s2)
                                {
                                        b=y;
                                        break;
                                }
                                y++;
                         }
```

```
                    mfterm[a][b]=1;
                }
        }
        cout<<"\n Firstterm :";
        display(mfterm);
}
void opm::lterm()
{
        char l1,l2;
        int x=0,y=0,a=0,b=0;
        int z,flag1=0;
        flushall();
        for(k=0;k<cp;k++)
        {
                flag1=0;
                if(strlen(rval[k])==1)
                {
                        for(i=0;i<nt;i++)
                        {
                                if(rval[k][0]==cnt[i])
                                {
                                        flag1=1;
                                        break;
                                }
                                else
                                        flag1=0;
                        }
                }
                if(flag1!=1)
                {
                        l1=lval[k][0];
                        z=strlen(rval[k]);
                        for(i=0;i<t;i++)
                        {
                          if(rval[k][z-1]==ct[i])
                                        l2=rval[k][z-1];
                          else if(rval[k][z-2]==ct[i])
                                        l2=rval[k][z-2];
                        }
                        x=y=0;
                        while(s[x]!='\0')
                        {
                                if(s[x]==l1)
                                {
                                        a=x;
                                        break;
                                }//if
                                x++;
                        }//while
                        while(s[y]!='\0')
```

```
            {
                if(s[y]==l2)
                {
                    b=y;
                    break;
                }
                y++;
            } //while
             mlterm[a][b]=1;
        }//if
    }//for
    cout<<"Lastterm :\n";
    display(mlterm);
}
void opm::last()
{
    char l1,l2;
    int x=0,y=0,a=0,b=0;
    int z;
    int n=t+nt;
    flushall();
    for(k=0;k<cp;k++)
    {
        l1=lval[k][0];
        z=strlen(rval[k]);
        l2=rval[k][z-1];
        x=y=0;
        while(s[x]!='\0')
        {
            if(s[x]==l1)
            {
                a=x;
                break;
            }//if
            x++;
        }//while
        while(s[y]!='\0')
        {
            if(s[y]==l2)
            {
                b=y;
                break;
            }
            y++;
        } //while
         ml[a][b]=1;
    }//for
    int i1=0;
    while(i1<n)
    {
```

```
                for(int j1=0;j1<n;j1++)
                {
                        if(ml[j1][i1]==1)
                        {
                                for(int k1=0;k1<n;k1++)
                                        ml[j1][k1]=(ml[j1][k1] || ml[i1][k1]);
                        }//if
                }
                i1++;
        }
        for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                        ml[i][i]=1;
        }
        cout<<"Last * :\n";
        display(ml);
}
void opm::greater()
{
        int n=t+nt;
        for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                        m3[i][j]=0;
        multiply(ml,mlterm);
        for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                        ml[i][j]=0;
        for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                        ml[j][i]=m3[i][j];
        }
        for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                        m3[i][j]=0;
        multiply(ml,meq);
        for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                        mltr[i][j]=m3[i][j];
        cout<<"\nGreater Matrix : ";
        display(mltr);
}
void opm::less()
{
        int n=t+nt;
        multiply(meq,mfstar);
        for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                        mfstar[i][j]=0;
        for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                        mfstar[i][j]=m3[i][j];
        }
        for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                        m3[i][j]=0;
        multiply(mfstar,mfterm);
        for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                        mfstar[i][j]=0;
        }
        for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                        mfstar[i][j]=m3[i][j];
        }
        cout<<"\n Less Matrix :";
        display(mfstar);
}

void opm::pequal()
{
        int i1,j1,i2,i3,k1,x,y,a,b;
        char l,m;
        for(j1=0;j1<cp;j1++)
        {
                for(k1=0;k1<strlen(rval[j1]);k1++)
                {
                  l=m='\0';
                  a=b=0;
                  for(i1=0;i1<t;i1++)
                        if(rval[j1][k1]==ct[i1])
                        {
                                l=rval[j1][k1];
                                for(i2=0;i2<t;i2++)
                                if(rval[j1][k1+1]==ct[i2])
                                {
                                        m=rval[j1][k1+1];
                                        break;
                                }
                                else if(rval[j1][k1+2]==ct[i2])
                                        for(i3=0;i3<t;i3++)
                                                if(rval[j1][k1+2]==ct[i3])
                                                {
                                                        m=rval[j1][k1+2];
                                                        break;
                                                }
```

```
            }//if
            if(l!='\0' && m!='\0')
            {
                x=y=0;
                while(s[x]!='\0')
                {
                    if(s[x]==l)
                    {
                        a=x;
                        break;
                    }//if
                    x++;
                }//while
                while(s[y]!='\0')
                {
                    if(s[y]==m)
                    {
                        b=y;
                        break;
                    }
                    y++;
                } //while
                eq[a][b]=1;
                a=b=0;
            }//if

        }
    }
    cout<<"\nEqual precedence:";
    display(eq);
}

void opm::multiply(int m1[10][10],int m2[10][10])
{
    int n=t+nt;
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            for(int k=0;k<n;k++)
                m3[i][j]=m3[i][j] + m1[i][k] * m2[k][j];
            if(m3[i][j]>=2)
                m3[i][j]=1;
        }
    }
}

void opm::simpose()
{
    int a,b,c;
    a=nt+t;
    for(i=0;i<a;i++)
        for(j=0;j<a;j++)
        {
            sim[i][j]='0';
            m[i][j]=0;
        }
    for(i=a-t;i<a;i++)
    {
        m[i][a-1]=3;
        m[a-1][i]=2;
    }
    for(i=0;i<a;i++)
    {
        for(j=0;j<a;j++)
        {
            if(mfstar[i][j]==1)
            {
                sim[i][j]='<';
                m[i][j]=2;
            }
            if(mltr[i][j]==1)
            {
                sim[i][j]='>';
                m[i][j]=3;
            }
            if(eq[i][j]==1)
            {
                sim[i][j]='=';
                m[i][j]=1;
            }
        }
    }
    display(m);
    cout<<"\nSuperimposed matrix:";
    cout<<"\n";
    for(i=0;i<a;i++)
        cout<<"\t"<<s[i];
    cout<<"\n ------------------------------------------------------------";
    for(i=0;i<a;i++)
    {
        cout<<"\n   "<<s[i];
        cout<<" |";
        for(j=0;j<a;j++)
            cout<<"\t"<<sim[i][j];
    }
}
void opm::parse()
{
    char c;
```

```
int q=0,m1=0,k=0,v,w;
char p[15],hand[10];
int a,b,fh=-1,bh=-1,inh=0,e=1;
int flag=0;
cout<<"\nEnter the string to be parsed:";
gets(p);
cout<<"\n Enter the exclusive symbol :";
cin>>c;
q=strlen(p);
m1=0;
while(m1<q)
{
  for(k=0;k<t;k++)
        if(p[m1]==ct[k])
                v=w=0;
                if(flag==0)
                {
                        a=b=0;
                        while(s[v]!='\0')
                        {
                                if(s[v]==p[m1])
                                {
                                        a=v;
                                        flag=1;
                                        break;
                                }
                         v++;
                        }
                }
                if(flag==1)
                {
                        m1++;
                        for(k=0;k<t;k++)
                                if(p[m1]==ct[k])
                                {
                                        while(s[w]!='\0')
                                        {
                                                if(s[w]==p[m1])
                                                {
                                                        b=w;
                                                        flag=0;
                                                        break;
                                                }
                                        w++;
                                        }//while
                                }//if
                }//if
        if(a!=0 && b!=0)
        {
                switch(m[a][b])
        {
            case 2:

              {
                 for(int k1=0;k1<nt;k1++)
                 {
                 if(p[m1-1]==cnt[k1])
                 {
                    fh=m1-1;
                    break;
                 }
                 else
                 {
                    fh=m1;
                  }
             }
                    break;
              }
            case 3: bh=m1-1; break;
            case 1: e++;
        }//switch
        if(fh>=0 && bh>0)
        {
            inh=fh;
            for(k=0;fh<=bh;fh++,k++)
                hand[k]=p[fh];
            hand[k]='\0';
            cout<<"\nHandle:"<<hand;
            for(k=0;k<cp;k++)
            {
                if(strcmp(rval[k],hand)==0)
                {
                    p[inh]=lval[k][0];
                    break;
                }
            }
            for(;p[bh]!='\0';)
                p[++inh]=p[++bh];
            p[bh]='\0';
            a=b=0;
            cout<<"\n P:"<<p;
            fh=bh=e=m1=-1;
        }//if
    }//if a,b
    if(a==0)
        m1++;
    if(p[1]==c &&strlen(p)==3)
        break;
}//while
    for(k=0;k<cp;k++)
```

```
            if(strcmp(rval[k],hand)==0)
                break;
            if(c==lval[k][0])
                cout<<"\nString is parsable"<<lval[k][0];
            else
                cout<<"\nString is not parsable";
}
void opm::display(int m[10][10])
{
    cout<<"\n\n";
    flushall();
    int p=t+nt;
    for(i=0;i<p;i++)
        cout<<"\t"<<s[i];
    cout<<"\n  ----------------------------------------------------------------";
    for(i=0;i<p;i++)
    {
        cout<<"\n   "<<s[i];
        cout<<" |";
        for(j=0;j<p;j++)
            cout<<"\t"<<m[i][j];
    }
}
void main()
{
    opm o1;
    int a;
    clrscr();
    flushall();
    o1.get_data();
    o1.matrix();
    o1.equal();
    getch();
    clrscr();
    o1.first();
    getch();
    clrscr();
    o1.last();
    getch();
    clrscr();
    o1.fterm();
    getch();
    clrscr();
    o1.lterm();
    getch();
    clrscr();
    o1.less();
    getch();
    clrscr();
    o1.greater();
```

```
            getch();
            clrscr();
            o1.pequal();
            getch();
            clrscr();
            o1.simpose();
            getch();
            clrscr();
            o1.parse();
            getch();
}
```

OUTPUT:

```
How many nonterminals:3
Enter nonterminals:ETF
How many terminals:5
Enter terminals:+*()i
How many productions:6
Enter the production:E=E+T
Enter the production:T=T*F
Enter the production:E=T
Enter the production:T=F
Enter the production:F=(E)
Enter the production:F=i
```

Program Output:

Equal :

```
     E    T    F    +    *    (    )    i    #
 --------------------------------------------------
 E | 0    0    0    1    0    0    1    0    0
 T | 0    0    0    0    1    0    0    0    0
 F | 0    0    0    0    0    0    0    0    0
 + | 0    1    0    0    0    0    0    0    0
 * | 0    0    1    0    0    0    0    0    0
 ( | 1    0    0    0    0    0    0    0    0
 ) | 0    0    0    0    0    0    0    0    0
 i | 0    0    0    0    0    0    0    0    0
 # | 0    0    0    0    0    0    0    0    0
```

First* :

```
     E    T    F    +    *    (    )    i    #
 --------------------------------------------------
 E | 1    1    1    0    0    1    0    1    0
 T | 0    1    1    0    0    1    0    1    0
 F | 0    0    1    0    0    1    0    1    0
 + | 0    0    0    1    0    0    0    0    0
 * | 0    0    0    0    1    0    0    0    0
 ( | 0    0    0    0    0    1    0    0    0
 ) | 0    0    0    0    0    0    1    0    0
 i | 0    0    0    0    0    0    0    1    0
 # | 0    0    0    0    0    0    0    0    1
```

Last * :

```
     E    T    F    +    *    (    )    i    #
 -----------------------------------------------
 E | 1    1    1    0    0    0    1    1    0
 T | 0    1    1    0    0    0    1    1    0
 F | 0    0    1    0    0    0    1    1    0
 + | 0    0    0    1    0    0    0    0    0
 * | 0    0    0    0    1    0    0    0    0
 ( | 0    0    0    0    0    1    0    0    0
 ) | 0    0    0    0    0    0    1    0    0
 i | 0    0    0    0    0    0    0    1    0
 # | 0    0    0    0    0    0    0    0    1
```

Firstterm :

```
     E    T    F    +    *    (    )    i    #
 -----------------------------------------------
 E | 0    0    0    1    0    0    0    0    0
 T | 0    0    0    0    1    0    0    0    0
 F | 0    0    0    0    0    1    0    1    0
 + | 0    0    0    0    0    0    0    0    0
 * | 0    0    0    0    0    0    0    0    0
 ( | 0    0    0    0    0    0    0    0    0
 ) | 0    0    0    0    0    0    0    0    0
 i | 0    0    0    0    0    0    0    0    0
 # | 0    0    0    0    0    0    0    0    0
```

Lastterm :

```
     E    T    F    +    *    (    )    i    #
 -----------------------------------------------
 E | 0    0    0    1    0    0    0    0    0
 T | 0    0    0    0    1    0    0    0    0
 F | 0    0    0    0    0    0    1    1    0
 + | 0    0    0    0    0    0    0    0    0
 * | 0    0    0    0    0    0    0    0    0
 ( | 0    0    0    0    0    0    0    0    0
 ) | 0    0    0    0    0    0    0    0    0
 i | 0    0    0    0    0    0    0    0    0
 # | 0    0    0    0    0    0    0    0    0
```

Less Matrix :

```
     E    T    F    +    *    (    )    i    #
 -----------------------------------------------
 E |  0    0    0    0    0    0    0    0    0
 T |  0    0    0    0    0    0    0    0    0
 F |  0    0    0    0    0    0    0    0    0
 + |  0    0    0    0    1    1    0    1    0
 * |  0    0    0    0    0    1    0    1    0
 ( | 0    0    0    1    1    1    0    1    0
 ) | 0    0    0    0    0    0    0    0    0
 i |  0    0    0    0    0    0    0    0    0
 # |  0    0    0    0    0    0    0    0    0
```

Greater Matrix :

```
     E    T    F    +    *    (    )    i    #
 -----------------------------------------------------------------
 E |  0    0    0    0    0    0    0    0    0
 T |  0    0    0    0    0    0    0    0    0
 F |  0    0    0    0    0    0    0    0    0
 + |  0    0    0    1    0    0    1    0    0
 * |  0    0    0    1    1    0    1    0    0
 ( |  0    0    0    0    0    0    0    0    0
 ) |  0    0    0    1    1    0    1    0    0
 i |  0    0    0    1    1    0    1    0    0
 # |  0    0    0    0    0    0    0    0    0
```

Equal precedence:

```
     E    T    F    +    *    (    )    i    #
 -----------------------------------------------------------------
 E |  0    0    0    0    0    0    0    0    0
 T |  0    0    0    0    0    0    0    0    0
 F |  0    0    0    0    0    0    0    0    0
 + |  0    0    0    0    0    0    0    0    0
 *  |  0    0    0    0    0    0    0    0    0
 ( |  0    0    0    0    0    0    1    0    0
 ) |  0    0    0    0    0    0    0    0    0
 i |  0    0    0    0    0    0    0    0    0
 # |  0    0    0    0    0    0    0    0    0
```

```
     E    T    F    +    *    (    )    i    #
 -----------------------------------------------------------------
 E |  0    0    0    0    0    0    0    0    0
 T |  0    0    0    0    0    0    0    0    0
 F |  0    0    0    0    0    0    0    0    0
 + |  0    0    0    3    2    2    3    2    3
 * |  0    0    0    3    3    2    3    2    3
 ( | 0    0    0    2    2    2    1    2    3
 ) | 0    0    0    3    3    0    3    0    3
 i | 0    0    0    3    3    0    3    0    3
 # |  0    0    0    2    2    2    2    2    2
```

Superimposed matrix:

```
       E     T     F     +     *     (     )     i     #
 ------------------------------------------------------------
   E |  0     0     0     0     0     0     0     0     0
   T |  0     0     0     0     0     0     0     0     0
   F |  0     0     0     0     0     0     0     0     0
   + |  0     0     0     >     <     <     >     <     0
   * |  0     0     0     >     >     <     >     <     0
   ( |  0     0     0     <     <     <     =     <     0
   ) |  0     0     0     >     >     0     >     0     0
   i |  0     0     0     >     >     0     >     0     0
   # |  0     0     0     0     0     0     0     0     0

Enter the string to be parsed:#T*(E+T)#

 Enter the exclusive symbol :T

Handle:E+T
 P:#T*(E)#
Handle:(E)
 P:#T*F#
Handle:T*F
 P:#T#
String is parsableT
```

PRACTICAL 09

AIM: Write a code to generate the DAG for the input arithmetic expression.

SOURCE CODE:

```
#include<iostream.h>
#include<string.h>
#include<stdio.h>
#include<stdio.h>
#include<conio.h>
void main()
{
    char node[10][10],optr[10],left[10][10],right[10][10],add[20][10];
    int flag,j,k,isSame=0,lastEntryEmpty=0,opt=0,n,i;
    clrscr();
    cout<<"Enter the total no of address:"<<endl;
    cin>>n; //total no of three addr codes
    cout<<"Enter the addresses:-->"<<endl;
    for(i=0;i<n;i++)
        cin>>add[i];
    char op[]={'+','-','*','/'};
    for(i=0;i<n;i++)
    {
            flag=1;
        char *nodeIndex=strchr(add[i],'=');//finding index of =sign
        int index=nodeIndex-add[i];//getting index of = sign
            if(nodeIndex)//if = sign found
            {
                if((add[i][index-1]=='>')||(add[i][index-1]=='<'))
                    flag=0;//whether it is <= or >= operator
                else    flag=1;//only = sign
            }
            if(flag==1)//
            {
                char *lhs,*rhs,*lNode,*rNode,*opIndex;;
                lhs=strtok(add[i],"=");//lhs of code
                rhs=strtok(NULL,add[i]);//rhs of code
                int isOp=0;
            for(k=0;k<strlen(rhs);k++)//for searching operator in rhs of code
                {
                    opIndex=strchr(op,rhs[k]);//searching optr
                    if(opIndex)//if optr
                    {
                        if(lastEntryEmpty==0)
                        {
                            optr[i]=rhs[k];//copy optr
                            strcpy(node[i],lhs);//copy node
                        }

                        else
                        {
                            optr[i-1]=rhs[k];//copy optr
                            strcpy(node[i-1],lhs);//copy node
                            opt++;
                        }
                        isOp=1;//setting flag
                    }
                    if(isOp==1)
                        break;
                    else
                        isOp=0;
                }
                if(isOp==1)
                {
                    char *opStr;//for optr as a string
                    sprintf(opStr,"%c",rhs[k]);
                    lNode=strtok(rhs,opStr);//for left child
                    rNode=strtok(NULL,opStr);//for rightchild
                    if(lastEntryEmpty==0)
                    {
                        strcpy(left[i],lNode);
                        strcpy(right[i],rNode);
                    }
                    else
                    {
                        strcpy(left[i-1],lNode);
                        strcpy(right[i-1],rNode);
                    }
                }
                else //if there no any operator
                {
                    for(j=0;j<i;j++)
                    {
                        if(strcmp(rhs,node[j])==0)
                            isSame=1;
                        else    isSame=0;
                        if(isSame==1)
                        {
                            strcat(node[j]," ");
                            strcat(node[j],lhs);
                            lastEntryEmpty=1;
                            if(lastEntryEmpty==1)
                                opt++;
                        }
                    } //end j
                }
            }
  }//end i
  cout<<"node \t optr\t left child \tright child"<<endl;
  for(i=0;i<=n-opt;i++)
      cout<<node[i]<<"\t"<<optr[i]<<"\t\t"<<left[i]<<"\t\t"<<right[i]<<endl;
 getch();
}
```