Practical Name: .....Practical No. .....

OBJECTIVE **Write a program that accepts a language containing even number of zeros and even number of ones**

CODE:

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
   string s;
   cout << "Enter the string" << endl;
   cin >> s;
   int countzero = 0, countones = 0;

   for (int i = 0; i < s.length(); i++)
   {
      if (s[i] == '0')
         countzero++;
      if (s[i] == '1')
         countones++;
   }
   if (countzero % 2 == 0 && countones % 2 == 0)
      cout << "String is accepted";
   else
      cout << "String is not accepted";
   return 0;
}
```

OUTPUT:

```
Enter the string
001100110
String is not accepted
```

```
Enter the string
00110000
String is accepted
```

Practical Name: .....Practical No. .....

OBJECTIVE Write a program that accepts a string which starts with 0 and ends with 11

CODE:

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
   string s;
   cout << "Enter the string" << endl;
   cin>>s;
   if(s[0]=='0'&& s[s.length()-1]=='1' &&  s[s.length()-2]=='1')
   cout<<"string is accepted";
   else
   cout<<"string is not accepted";

   return 0;
}
```

OUTPUT:

```
Enter the string
01110110
string is not accepted
```

```
Enter the string
01100000001110011
string is accepted
```

Practical Name: .....Practical No. .....

OBJECTIVE Write a program for modulo 3

CODE:

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
   string s;
   cout<<"Enter String"<<endl;
   cin>>s;
   int q=0;
   for (int i=0;i<s.length();i++)
   {
      if(q==0)
      {
         if(s[i]=='0')
         q=0;
         else
         q=1;
      }
      else if(q==1)
      {
         if(s[i]=='0')
         q=2;
         else
         q=0;
      }
      else
      {
         if(s[i]=='0')
         q=1;
         else
         q=2;
      }

   }
   cout<<"Remainder is "<< q;
   return 0;
}
```

OUTPUT:

```
Enter String
10110
Remainder is 1
```

```
Enter String
101
Remainder is 2
```

```
Enter String
011
Remainder is 0
```

OBJECTIVE Write a program for conversion of NFA to DFA.

CODE:

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

using namespace std;

std::string &removeDuplicate(std::string &str)
{
   int length = str.length();
   for (unsigned int i = 0; i < length; i++)
   {
      char currChar = str[i];
      for (unsigned int j = i + 1; j < length; j++)
      {
         if (currChar == str[j])
            str.erase(std::remove(str.begin() + j, str.end(), str[j]), str.end());
      }
   }
   return str;
}

void remove(std::vector<string> &v)
{
   auto end = v.end();
   for (auto it = v.begin(); it != end; ++it)
   {
      end = std::remove(it + 1, end, *it);
   }

   v.erase(end, v.end());
}

int main()
{
   int states, input_variables;
   cout << "Enter number of states" << endl;
   cin >> states;
   cout << "Enter number of input_variables" << endl;
   cin >> input_variables;
   map<char, vector<pair<int, string>>> mp;
   cout << "Enter NFA" << endl;
   for (int i = 0; i < states; i++)
   {
      for (int j = 0; j < input_variables; j++)
      {
         cout<<"Enter next state when current state is "<<i<<" and input variable is "<<j<<endl;
         string a;
         cin >> a;
         mp[(char)(i + 48)].push_back({j, a});
      }
   }

   vector<string> temp;
   // vector<string> vs;
   map<string, int> mps;
   auto it = mp.begin();
```

```
string s = "";
s = it->first;
mps[s] = 0;
temp.push_back(s);
for (int i = 0; i < input_variables; i++)
{
   mps[(it->second[i]).second] = 0;
   temp.push_back((it->second[i]).second);
}
map<string, vector<pair<int, string>>> ans;

int i = 0;
while (i < temp.size())
{

   if (mps[temp[i]] == 0)
   {

      mps[temp[i]] = 1;
      string first;
      for (int j = 0; j < input_variables; j++)
      {
         first = "";
         // cout << i.first << endl;
         for (int z = 0; z < (temp[i]).length(); z++) // 2  01
         {
            first = first + mp[temp[i][z]][j].second;
         }
         first = removeDuplicate(first);
         string new_s = "";
         for (int k = 0; k < first.length(); k++)
         {
            if (first[k] >= '0' && first[k] <= '9')
            {
               new_s = new_s + first[k];
            }
         }
         sort(new_s.begin(), new_s.end());
         if (!mps[new_s])
         {
            mps[new_s] = 0;
            temp.push_back(new_s);
         }
         if (new_s == "")
         {
            ans[temp[i]].push_back({j, "NA"});
            continue;
         }
         // new_s = new_s;
         ans[temp[i]].push_back({j, new_s});
      }
   }
   i++;
}

remove(temp);
cout << "CONVERTED DFA TRANSITION TABLE IS\n\n";
cout << "STATES"
    << "\t\t"
    << "INPUTS"
```

```
        << "\n";
    for (int i = 0; i < input_variables; i++)
        cout << " "
            << "\t\t" << i;
    cout << "\n";
    for (int i = 0; i < temp.size(); i++)
    {
        if (temp[i] != "")
        {
            cout << temp[i] << "\t\t";
            for (int j = 0; j < input_variables; j++)
                cout << ans[temp[i]][j].second << "\t\t";
            cout << endl;
        }
    }
    return 0;
}
```

OUTPUT:

```
Enter number of states
2
Enter number of input_variables
2
Enter NFA
Enter next state when current state is 0 and input variable is 0
0
Enter next state when current state is 0 and input variable is 1
1
Enter next state when current state is 1 and input variable is 0
1
Enter next state when current state is 1 and input variable is 1
01
CONVERTED DFA TRANSITION TABLE IS

STATES          INPUTS
                0               1
0               0               1
1               1               01
01              01              01
```