DATA STRUCTURES PRACTICAL

By-Harsh Meena

88028

Q1. Write a Program to create a SET **A** and determine the cardinality of SET for an input array of elements (repetition allowed) and perform the following operations on the SET:
a) ismember (a, A): check whether an element belongs to set or not and return value as true/false.
b) powerset(A): list all the elements of power set of A.

A1 –

```
#include <iostream>
#include <set>
#include<math.h>
using namespace std;
int isMember(multiset<int> A){
  cout<<"\nEnter an element to find in the set\n";
  int c;
  cin>>c;
  if(A.count(c))
    cout<<"Element present\n";
  else
    cout<<"Sorry no such element present\n";
}
int powerSet(multiset<int> A){
  for(int counter = 0; counter < pow(2,A.size()); counter++)
  {
  for(int j = 0; j < A.size(); j++)
  {
    if(counter & (1 << j)){
      auto first = A.begin();
      std::advance(first, j);
      cout << *first;
    }
  }
  cout << endl;
  }
}
int dispSet(multiset<int> A){
  cout<<"Entered set is:-\n";
  for (auto it = A.begin(); it != A.end(); ++it)
    cout <<" "<< *it;
  cout<<endl;
  cout<<"Cardinality of entered set is:";
  cout<<A.size();
  return 0;
}
```

```
int inputSet(multiset<int> A){
   int c=1;
   cout<<"Enter the elements in the set-:\n";
   while(c){
      int n;
      cin>>n;
      A.insert(n);
      cout<<"do you want to enter more?<0/1>\n";
      cin>>c;
   }
   dispSet(A);
   isMember(A);
   powerSet(A);
   return 0;
}
int main(){
   multiset<int> A;
   inputSet(A);
   return 0;
}
```

Q2. Create a class SET and take two sets as input from user to perform following SET Operations:

a) Subset: Check whether one set is a subset of other or not.

b) Union and Intersection of two Sets.

c) Complement: Assume Universal Set as per the input elements from the user.

d) Set Difference and Symmetric Difference between two SETS

e) Cartesian Product of Sets.

A2 –

```
#include <bits/stdc++.h>
using namespace std;

class SET
{
      int size;
      char set[100];

public:
      SET(int s)
      {
            size = s;
            set[size];
      }

      void input()
      {
            cout << "Enter the elements of set:-   " << endl;
```

```
    for (int i = 0; i < size; i++)
    {
        cin >> set[i];
    }
}

bool isSubSet(SET set1)
{
    bool table[set1.size] = {false};
    for (int i = 0; i < set1.size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            if (set1.set[i] == set[j])
            {
                table[i] = true;
                break;
            }
        }
    }

    for (int i = 0; i < set1.size; i++)
    {
        if (table[i] == false)
        {
            return false;
        }
    }

    return true;
}

void Union(SET set1)
{
    vector<char> setunion;

    for (int i = 0; i < size; i++)
    {
        setunion.push_back(set[i]);
    }

    bool table[set1.size] = {false};
    for (int i = 0; i < set1.size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            if (set1.set[i] == set[j])
            {
                table[i] = true;
                break;
            }
            }
        }

        for (int i = 0; i < set1.size; i++)
        {
            if (table[i] == false)
            {
                setunion.push_back(set1.set[i]);
            }
        }

        cout << "Union of both sets is:- " << endl
             << "{";
        for (int i = 0; i < setunion.size(); i++)
        {
            cout << setunion[i] << ",";
        }
        cout << "}" << endl;
    }

    void Intersection(SET set1)
    {
        vector<char> setintersection;

        bool table[set1.size] = {false};
        for (int i = 0; i < set1.size; i++)
        {
            for (int j = 0; j < size; j++)
            {
                if (set1.set[i] == set[j])
                {
                    table[i] = true;
                    break;
                }
            }
        }

        for (int i = 0; i < set1.size; i++)
        {
            if (table[i] == true)
            {
                setintersection.push_back(set1.set[i]);
            }
        }

        cout << "Intersection of both sets is:- " << endl
             << "{";
        for (int i = 0; i < setintersection.size(); i++)
        {
            cout << setintersection[i] << ",";
        }
        cout << "}" << endl;
    }

    void Complement()
    {
        vector<bool> checkvalue(3);
        checkvalue.push_back(false);
        checkvalue.push_back(false);
        checkvalue.push_back(false);
        vector<char> universalset;

        for (int i = 0; i < size; i++)
        {
            if (set[i] >= 'A' && set[i] <= 'Z')
            {
                checkvalue[0] = true;
            }
            else if (set[i] >= 'a' && set[i] <= 'z')
            {
                checkvalue[1] = true;
            }
            else if (set[i] >= '0' && set[i] <= '9')
            {
                checkvalue[2] = true;
            }
        }

        if (checkvalue[0] == true)
        {
            for (int i = 0; i < 26; i++)
            {
                universalset.push_back(char(i + 65));
            }
        }
        if (checkvalue[1] == true)
        {
            for (int i = 0; i < 26; i++)
            {
                universalset.push_back(char(i + 97));
            }
        }
        if (checkvalue[2] == true)
        {
            for (int i = 0; i < 10; i++)
            {
                universalset.push_back(char(i + 48));
            }
        }

        for (int i = 0; i < size; i++)
        {
            for (int j = 0; j < universalset.size(); j++)
            {
                if (set[i] == universalset[j])
                {
                    universalset.erase(universalset.begin() + j);
                    break;
                }
            }
        }

        cout << "{";
        for (int i = 0; i < universalset.size(); i++)
        {
            cout << universalset[i] << ",";
        }
        cout << "}" << endl
           << endl;
    }

    void SetDifference(SET set1)
    {
        vector<char> setdifference;

        vector<char> setintersection;

        bool table[set1.size] = {false};
        for (int i = 0; i < set1.size; i++)
        {
            for (int j = 0; j < size; j++)
            {
                if (set1.set[i] == set[j])
                {
                    table[i] = true;
                    break;
                }
            }
        }

        for (int i = 0; i < set1.size; i++)
        {
            if (table[i] == true)
            {
                setintersection.push_back(set1.set[i]);
            }
        }

        for (int i = 0; i < size; i++)
        {
            bool flag = false;
            for (int j = 0; j < setintersection.size(); j++)
            {
                if (set[i] == setintersection[j])
                {
                    flag = false;
                    break;
                }
                else
                {
                    flag = true;
                }
            }
            if (flag == true)
            {
                setdifference.push_back(set[i]);
            }
        }

        cout << "{";
        for (int i = 0; i < setdifference.size(); i++)
        {
            cout << setdifference[i] << ",";
        }
        cout << "}" << endl
           << endl;
    }

    void SymmetricDifference(SET set1)
    {
        vector<char> symdifference;

        vector<char> setunion;

        for (int i = 0; i < size; i++)
        {
            setunion.push_back(set[i]);
        }

        bool table[set1.size] = {false};
        for (int i = 0; i < set1.size; i++)
        {
            for (int j = 0; j < size; j++)
            {
                if (set1.set[i] == set[j])
                {
                    table[i] = true;
                    break;
                }
            }
        }

        for (int i = 0; i < set1.size; i++)
        {
```

```
    if (table[i] == false)
    {
        setunion.push_back(set1.set[i]);
    }
}

for(int i=0;i<setunion.size();i++)
{
    symdifference.push_back(setunion[i]);
}

vector<char> setintersection;

bool table1[set1.size] = {false};
for (int i = 0; i < set1.size; i++)
{
    for (int j = 0; j < size; j++)
    {
        if (set1.set[i] == set[j])
        {
            table1[i] = true;
            break;
        }
    }
}

for (int i = 0; i < set1.size; i++)
{
    if (table1[i] == true)
    {
        setintersection.push_back(set1.set[i]);
    }
}

for (int i = 0; i < symdifference.size(); i++)
{
    bool flag = false;
    for (int j = 0; j < setintersection.size(); j++)
    {
        if (symdifference[i] == setintersection[j])
        {
            symdifference.erase(symdifference.begin() + i);
        }
    }
}

cout << "{";
for (int i = 0; i < symdifference.size(); i++)
{
    cout << symdifference[i] << ",";
}
```

```
        cout << "}" << endl
            << endl;
    }

    void CartesianProd(SET set1)
    {
        string cartprod[set1.size][size];

        for(int i=0;i<size;i++)
        {
            for(int j=0;j<set1.size;j++)
            {
                string temp = "(";
                temp = temp + set[i];
                temp = temp + ",";
                temp = temp + set1.set[j];
                temp = temp + ")";

                cartprod[j][i] = temp;
            }
        }

        cout<<"{";
        for(int i=0;i<size;i++)
        {
            for(int j=0;j<set1.size;j++)
            {
                cout<<cartprod[j][i]<<",";
            }
        }
        cout<<"}";
    }
};

int main()
{
    int size;

    cout << "Enter size of first set:-   ";
    cin >> size;

    SET set1(size);
    set1.input();

    cout << "Enter size of second set:-   ";
    cin >> size;

    SET set2(size);
    set2.input();

    bool issubset = set1.isSubSet(set2);
    if (issubset)
    {
        cout << "Set 2 is subset of Set 1" << endl;
    }
    else
    {
        cout << "Set 2 is not a subset of Set 1" << endl;
    }

    set1.Union(set2);

    set1.Intersection(set2);

    cout << "Complement of first set:-   " << endl;
    set1.Complement();

    cout << "Complement of second set:-   " << endl;
    set2.Complement();

    cout << "Set1 - Set2" << endl;
    set1.SetDifference(set2);

    cout << "Set2 - Set1" << endl;
    set2.SetDifference(set1);

    cout << "Symmetric difference" << endl;
    set1.SymmetricDifference(set2);

    cout<<"Cartesian product of both the sets:-  "<<endl;
    set1.CartesianProd(set2);

    return 0;
}
```

Q3. Create a class RELATION, use Matrix notation to represent a relation. Include functions to check if a relation is reflexive, Symmetric, Anti-symmetric and Transitive. Write a Program to use this class.

Q4. Use the functions defined in Ques 3 to find check whether the given relation is:

a) Equivalent, or

b) Partial Order relation, or

c) None

A –

```
#include<iostream>
using namespace std;
class Relation{
   public:
   int f=0;
   int rel[4][4] ={{1,1,1},
                  {1,1,1},
            {1,1,1}
                };
int checkRef(){
   for(int i=0;i<3;i++){
      if(rel[i][0]!=1){
         cout<<"Not reflexive ";
         return 0;
      }
   }
   cout<<"Reflexive ";
   f++;
   return 0;
}
int checkSym(){
   for(int i=0;i<3;i++){
      for(int j=0;j<3;j++){
         if(rel[i][j]==1 && rel[j][i]!=1){
           cout<<"Not symmetric ";
           return 0;
         }
      }
   }
   cout<<"Symmetric ";
   f++;
   return 0;
}
int checkAntiSym(){
   for(int i=0;i<3;i++){
      for(int j=0;j<3;j++){
         if(rel[i][j]==1 && rel[j][i]!=1){
           cout<<"Antisymmetric ";
           f++;
           return 0;
         }
      }
   }
   cout<<"Not Antisymmetric ";
   return 0;
}
int checkTrans(){
   for(int i=0;i<3;i++){
      for(int j=0;j<3;j++){
         for(int k=0;k<3;k++){
            if(rel[i][j] && rel[j][k] && !rel[i][k]){
               cout<<"Not Transitive ";
               return 0;
            }
         }
      }
   }
   f++;
```

```
        cout<<"Transitive ";
        return 0;


    }
    int checkEqui(){
        checkRef();
        cout<<", ";
        checkSym();
        cout<<"and ";
        checkTrans();
        if(f==3){
            cout<<"so in conclusion the relation is an Equivalence relation";
        }
        else{
            cout<<"so in conclusion the relation is not an Equivalence relation";
        }
    }
    int checkParOrder(){
        checkRef();
        cout<<", ";
        checkAntiSym();
        cout<<"and ";
        checkTrans();
        if(f==3){
            cout<<"so in conclusion the relation is a Partial order relation";
        }
        else{
            cout<<"so in conclusion the relation is not a Partial order relation";
        }

    }
};
int main(){
     Relation r;
     int n;
     while(n!=7){
     cout<<"\nWhat do you want to do with this relation?\n";
     cout<<"1.Check Symmetry\n";
     cout<<"2.Check Reflexivity\n";
     cout<<"3.Check Antisymmetry\n";
     cout<<"4.Check Transitivity\n";
     cout<<"5.Check Equivalence\n";
     cout<<"6.Check Partial Order\n";
     cout<<"7.Exit\n";
     cin>>n;
     switch(n){
     case(1):
        cout<<"The relation is ";
        r.checkSym();
        break;
     case(2):
        cout<<"The relation is ";
        r.checkRef();
        break;
     case(3):
        cout<<"The relation is ";
        r.checkAntiSym();
        break;
     case(4):
        cout<<"The relation is ";
        r.checkTrans();
        break;
     case(5):
        cout<<"The relation is ";
        r.checkEqui();
        break;
     case(6):
        cout<<"The relation is ";
        r.checkParOrder();
        break;
     case(7):
        cout<<"Bye!";
        break;
     default:
        cout<<"Please enter a number between 1-5\n";
     }
     }

}
```

Q13 - Write a Program to accept the truth values of variables **x** and **y**, and print the truth table of the following logical operations:

a) Conjunction

b) Disjunction

c) Exclusive OR

d) Conditional

e) Bi-conditional

f) Exclusive NOR

g) Negation

h) NAND

i) NOR

A13 –

```
#include<iostream>
#include<set>
#include<stdio.h>
#include<iomanip>
using namespace std;
int exOR(){
   cout<<setw(5)<<"x"<<setw(5)<<"|"<<setw(5)<<"y"<<setw(5)<<"|"<<setw(10)<<"x XOR y\n";
   cout<<"   -------------------------\n";
   for(int i=0;i<=1;i++){
      for(int j=0;j<=1;j++){
            cout<<setw(5)<<i<<setw(5)<<"|"<<setw(5)<<j<<setw(5)<<"|";
            cout<<"    ";
            printf("%d",i ^ j);
            cout<<"\n";

        }
    }
    return 0;
}
int conj(){
    cout<<setw(5)<<"x"<<setw(5)<<"|"<<setw(5)<<"y"<<setw(5)<<"|"<<setw(10)<<"x AND y\n";
    cout<<"   -------------------------\n";
    for(int i=0;i<=1;i++){
        for(int j=0;j<=1;j++){
            cout<<setw(5)<<i<<setw(5)<<"|"<<setw(5)<<j<<setw(5)<<"|";
            cout<<"    ";
            printf("%d",i && j);
            cout<<"\n";

        }
    }
    return 0;
}
int disj(){
    cout<<setw(5)<<"x"<<setw(5)<<"|"<<setw(5)<<"y"<<setw(5)<<"|"<<setw(10)<<"x OR y\n";
    cout<<"   -------------------------\n";
    for(int i=0;i<=1;i++){
        for(int j=0;j<=1;j++){
            cout<<setw(5)<<i<<setw(5)<<"|"<<setw(5)<<j<<setw(5)<<"|";
            cout<<"    ";
            printf("%d",i || j);
            cout<<"\n";

        }
    }
    return 0;
}
int cond(){
    cout<<setw(5)<<"x"<<setw(5)<<"|"<<setw(5)<<"y"<<setw(5)<<"|"<<setw(10)<<"x --> y\n";
    cout<<"   -------------------------\n";
    for(int i=0;i<=1;i++){
        for(int j=0;j<=1;j++){
            cout<<setw(5)<<i<<setw(5)<<"|"<<setw(5)<<j<<setw(5)<<"|";
            cout<<"    ";
            printf("%d",!i || j);
            cout<<"\n";

        }
    }
    return 0;
}
```

```
int biCond(){
   cout<<setw(5)<<"x"<<setw(5)<<"|"<<setw(5)<<"y"<<setw(5)<<"|"<<setw(10)<<"x <--> y\n";
   cout<<"   -------------------------\n";
   for(int i=0;i<=1;i++){
      for(int j=0;j<=1;j++){
            cout<<setw(5)<<i<<setw(5)<<"|"<<setw(5)<<j<<setw(5)<<"|";
            cout<<"    ";
            printf("%d",((!i || j) && (!j || i)));
            cout<<"\n";

      }
   }
   return 0;
}
int exNOR(){
   cout<<setw(5)<<"x"<<setw(5)<<"|"<<setw(5)<<"y"<<setw(5)<<"|"<<setw(10)<<"x XOR y\n";
   cout<<"   -------------------------\n";
   for(int i=0;i<=1;i++){
      for(int j=0;j<=1;j++){
            cout<<setw(5)<<i<<setw(5)<<"|"<<setw(5)<<j<<setw(5)<<"|";
            cout<<"    ";
            printf("%d",!(i ^ j));
            cout<<"\n";

      }
   }
   return 0;
}
int neg(){
   cout<<setw(5)<<"x"<<setw(5)<<"|"<<setw(7)<<"x'\n";
   cout<<"   -------------\n";
   for(int i=0;i<=1;i++){
      cout<<setw(5)<<i<<setw(5)<<"|";
         cout<<"    ";
         printf("%d",!i);
         cout<<"\n";
   }
   return 0;

}
int nand(){
   cout<<setw(5)<<"x"<<setw(5)<<"|"<<setw(5)<<"y"<<setw(5)<<"|"<<setw(10)<<"x NAND y\n";
   cout<<"   -------------------------\n";
   for(int i=0;i<=1;i++){
      for(int j=0;j<=1;j++){
            cout<<setw(5)<<i<<setw(5)<<"|"<<setw(5)<<j<<setw(5)<<"|";
            cout<<"    ";
            printf("%d",!(i && j));
            cout<<"\n";

      }
   }
   return 0;

}
int nor(){
   cout<<setw(5)<<"x"<<setw(5)<<"|"<<setw(5)<<"y"<<setw(5)<<"|"<<setw(10)<<"x NOR y\n";
   cout<<"   -------------------------\n";
   for(int i=0;i<=1;i++){
      for(int j=0;j<=1;j++){
            cout<<setw(5)<<i<<setw(5)<<"|"<<setw(5)<<j<<setw(5)<<"|";
            cout<<"    ";
            printf("%d",!(i || j));
            cout<<"\n";

      }
   }
   return 0;

}
int main(){
   conj();
   cout<<"\n\n";
   disj();
   cout<<"\n\n";
   exOR();
   cout<<"\n\n";
   cond();
   cout<<"\n\n";
   biCond();
   cout<<"\n\n";
   exNOR();
   cout<<"\n\n";
   neg();
   cout<<"\n\n";
   nand();
   cout<<"\n\n";
   nor();

}
```

Q14 - Write a program to accept an input **n** from the user and graphically represent the values of **T(n)** where **n** varies from **0** to **n** for the recurrence relations. For e.g. T(n) = T(n-1) + n, T(0) = 1, T(n) = T(n-1) + n^2, T(0) =1, T(n) = 2*T(n)/2 + n, T(1)=1.

A14 –

```
#include <iostream>
using namespace std;
int recurrenceOne(int n)
{
    if (n == 0)
        return 1;
    return recurrenceOne(n - 1) + n;
}
int recurrenceTwo(int n)
{
    if (n == 0)
        return 1;
    return recurrenceTwo(n - 1) + n * n;
}
int recurrenceThree(int n)
{
    if (n == 1)
        return 1;
    return 2 * recurrenceThree(n / 2) + n;
}

int main()
{
    int n, ch;
    cout << "Choose Recurrence Relation to Evaluate:\n"
         << "  (1) T(n) = T(n - 1) + n and T(0) = 1\n"
         << "  (2) T(n) = T(n - 1) + n^2 and T(0) = 1\n"
         << "  (3) T(n) = 2 * T(n / 2) + n and T(1) = 1\n";
    cout << "Enter Choice: ";
    cin >> ch;
    switch (ch)
    {
    case 1:
        cout << "\nEnter Value of n: ";
        cin >> n;
        cout << "\nValues for T(n) = T(n - 1) + n:\n";
        for (int i = n; i >= 0; i--)
        {
            if (i == 0)
                cout << "T(0) = " << recurrenceOne(i)
                     << endl;
            else
                cout << "T(" << i << ") = T(" << (i - 1)
                     << ") + " << i << " = "
                     << recurrenceOne(i)
                     << endl;
        }
        break;
    case 2:
        cout << "\nEnter Value of n: ";
        cin >> n;
        cout << "\nValues for T(n) = T(n - 1) + n^2:\n";
        for (int i = n; i >= 0; i--)
        {
            if (i == 0)
```

```
                cout << "T(0) = " << recurrenceTwo(i)
                    << endl;
            else
                cout << "T(" << i << ") = T(" << (i - 1)
                    << ") + " << i * i << " = "
                    << recurrenceTwo(i)
                    << endl;
        }
        break;
    case 3:
        cout << "\nEnter Value of n: ";
        cin >> n;
        cout << "\nValues for T(n) = 2 * T(n / 2) + n:\n";
        for (int i = n; i >= 1; i--)
        {
            if (i == 1)
                cout << "T(1) = " << recurrenceThree(i)
                    << endl;
            else
                cout << "T(" << i << ") = 2 * T(" << i
                    << " / 2) + " << i << " = "
                    << "2 * T(" << (i / 2)
                    << ") + " << i << " = "
                    << recurrenceThree(i)
                    << endl;
        }
        break;
    default:
        cout << "\nInvalid Choice!\n";
        break;
    }
    return 0;
}
```