ATMA RAM SANATAN DHARMA COLLEGE

Discrete Mathematical structures

Name - Shishirant Singh
Course - B.Sc(HONS.) Computer science
Roll No. – 28081

PRACTICAL 1

Create a class SET. Create member functions to perform the following SET operations:
1) ismember: check whether an element belongs to the set or not and return value as true/false.
2) powerset: list all the elements of the power set of a set.
3) subset: Check whether one set is a subset of the other or not.
4) union and Intersection of two Sets.
5) complement: Assume Universal Set as per the input elements from the user.
6) set Difference and Symmetric Difference between two sets.
7) cartesian Product of Sets.
Write a menu driven program to perform the above functions on an instance of the SET Class.

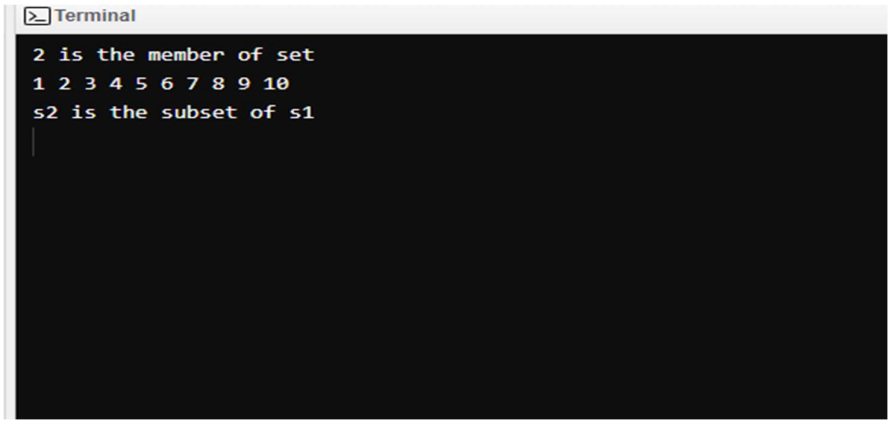
ANSWER:

```
#include<iostream>
#include<set>
#include<algorithm>
#include<cmath>
using namespace std;
class SET{
 public:
 set <int> s1={1,2,3,4,5};
 set <int> s2={2,3,4};
 void ismember(int i){
 if(s1.count(i)>0){
 cout<<i<<" is the member of set\n";
 }
 else{
 cout<<i<<" is not the memeber of set\n";
 }
 }
 void unionof(){
 set <int> s2={6,7,8,9,10};
 set <int> s3;

set_union(s1.begin(),s1.end(),s2.begin(),s2.end(),insert
er(s3,s3.begin()));
 for (auto i:s3){
 cout<<i<<" ";
 }
 cout<<endl;
 }
 void is_subset(){
 bool
issubset=includes(s1.begin(),s1.end(),s2.begin(),s2.end(
));
```

```
 if(issubset){
 cout<<"s2 is the subset of s1\n";
 }
 else{
 cout<<"s2 is not subset of s1\n";
 }
 }
 void set_diff(){
 set <int> s3;

set_difference(s1.begin(),s2.end(),s2.begin(),s2.end(),i
nserter(s3,s3.begin()));
 for (auto i:s3){cout<<i<<" ";}
 cout<<endl;
 }
 void symmetric_diff(){
 set <int> s3;

set_symmetric_difference(s1.begin(),s2.end(),s2.begin(),
s2.end(),inserter(s3,s3.begin()));
 for (auto i:s3){cout<<i<<" ";}
 cout<<endl;
 }
 void powerset(){
 int s1[]={1,2,3};
 int size=sizeof(s1)/sizeof(s1[0]);
 int pset_size=pow(2,size);
 int i,j;
 for(int i=0;i<pset_size;i++){
 for(int j=0;j<size;j++){
 if(i&(i<<j)){
 cout<<s1[j];
 }
 }
 cout<<endl;

 }
 }
 void cartesian(){
 int arr1[]={1,2,3};
 int arr2[]={4,5,6};
 int n1=sizeof(arr1)/sizeof(arr1[0]);
 int n2=sizeof(arr2)/sizeof(arr2[0]);
 for(int i=0;i<n1;i++){
 for(int j=0;j<n2;j++){
 printf("{%d,%d}, ",arr1[1],arr2[j]);
 }
 }
```

```
 }
};
int main(){
 SET s4;
 s4.ismember(2);
 s4.unionof();
 s4.is_subset();
 s4.set_diff();
 s4.symmetric_diff();
 s4.powerset();
 s4.cartesian();
 return 0;
}
```

OUTPUT:



PRACTICAL 2

Create a class RELATION, use Matrix notation to represent a relation. Include member functions to check if the relation is Reflexive, Symmetric, Anti-symmetric, Transitive. Using these functions check whether the given relation is: Equivalence or Partial Order relation or None.

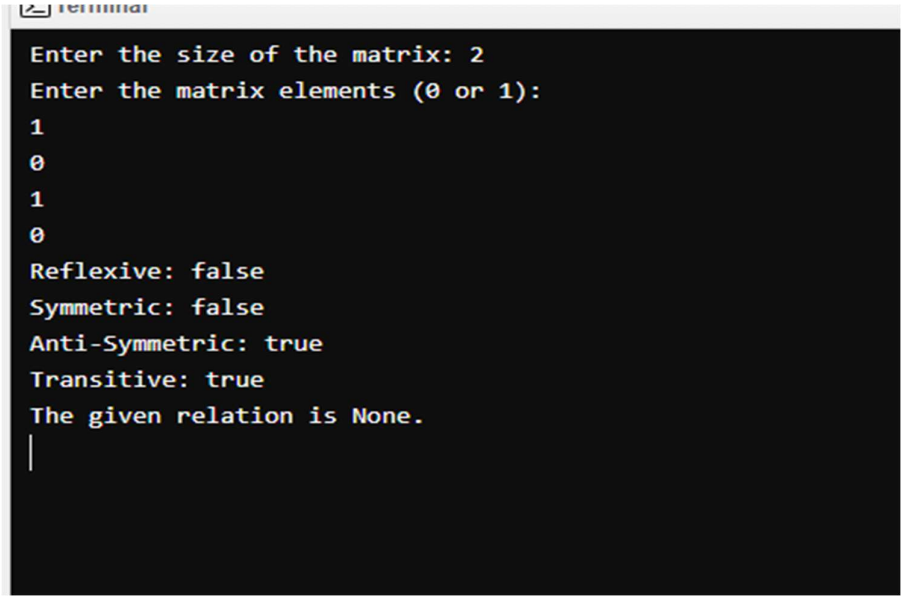
ANSWER:

```
#include <iostream>
#include <vector>
class RELATION {
private:
 std::vector<std::vector<bool>> matrix;
 int size;
public:
 RELATION(int n) : size(n) {
 matrix.resize(n, std::vector<bool>(n, false));
 }
 void setElement(int i, int j, bool value) {
 matrix[i][j] = value;
 }
 bool isReflexive() {
 for (int i = 0; i < size; ++i) {
 if (!matrix[i][i]) {
 return false;
 }
 }
 return true;
 }
 bool isSymmetric() {
 for (int i = 0; i < size; ++i) {
 for (int j = 0; j < size; ++j) {
 if (matrix[i][j] != matrix[j][i]) {
 return false;
 }
 }
 }
 return true;
 }
 bool isAntiSymmetric() {
 for (int i = 0; i < size; ++i) {
 for (int j = 0; j < size; ++j) {
 if (matrix[i][j] && matrix[j][i] && i !=
j) {
 return false;
 }
 }
 }
```

```
 return true;
 }
 bool isTransitive() {
 for (int i = 0; i < size; ++i) {
 for (int j = 0; j < size; ++j) {
 if (matrix[i][j]) {
 for (int k = 0; k < size; ++k) {
 if (matrix[j][k] &&
!matrix[i][k]) {
 return false;
 }
 }
 }
 }
 }
 return true;
 }
 void determineRelationType() {
 bool isEquivalence = isReflexive() &&
isSymmetric() && isTransitive();
 bool isPartialOrder = isReflexive() &&
isAntiSymmetric() && isTransitive();
 if (isEquivalence) {
 std::cout << "The given relation is an
Equivalence relation." << std::endl;
 } else if (isPartialOrder) {
 std::cout << "The given relation is a
Partial Order relation." << std::endl;
 } else {
 std::cout << "The given relation is None."
<< std::endl;
 }
 }
};
int main() {
 int n;
 std::cout << "Enter the size of the matrix: ";
 std::cin >> n;
 RELATION relation(n);
 std::cout << "Enter the matrix elements (0 or 1):"
<< std::endl;
 for (int i = 0; i < n; ++i) {
 for (int j = 0; j < n; ++j) {
 int value;
 std::cin >> value;
 relation.setElement(i, j, value);
 }
 }
```

```
 std::cout << std::boolalpha;
 std::cout << "Reflexive: " << relation.isReflexive()
<< std::endl;
 std::cout << "Symmetric: " << relation.isSymmetric()
<< std::endl;
 std::cout << "Anti-Symmetric: " <<
relation.isAntiSymmetric() << std::endl;
 std::cout << "Transitive: " <<
relation.isTransitive() << std::endl;
 relation.determineRelationType();
 return 0;
}
```

OUTPUT:



PRACTICAL 3

Write a Program that generates all the permutations of a given set of digits, with or without repetition.

ANSWER:

```
#include <iostream>
#include <vector>
void generatePermutations(std::vector<int>& digits,
std::vector<int>& permutation, std::vector<bool>&
chosen, bool withRepetition) {
 if (permutation.size() == digits.size()) {
 for (int digit : permutation) {
 std::cout << digit << " ";
 }
 std::cout << std::endl;
 } else {
 for (int i = 0; i < digits.size(); ++i) {
 if (!withRepetition && chosen[i]) {
 continue;
 }
 chosen[i] = true;
 permutation.push_back(digits[i]);
 generatePermutations(digits, permutation,
chosen, withRepetition);
 chosen[i] = false;
 permutation.pop_back();
 if (!withRepetition) {
 while (i + 1 < digits.size() &&
digits[i] == digits[i + 1]) {
 ++i;
 }
 }
 }
 }
}
int main() {
 int n;
 std::cout << "Enter the number of digits: ";
 std::cin >> n;
 std::vector<int> digits(n);
 std::cout << "Enter the digits: ";
 for (int i = 0; i < n; ++i) {
```

```
 std::cin >> digits[i];
 }
 bool withRepetition;
 std::cout << "Generate permutations with repetition? (1 for Yes, 0 for No): ";
 std::cin >> withRepetition;
 std::vector<int> permutation;
 std::vector<bool> chosen(n, false);
 generatePermutations(digits, permutation, chosen,
withRepetition);
 return 0;
}
```

OUTPUT:

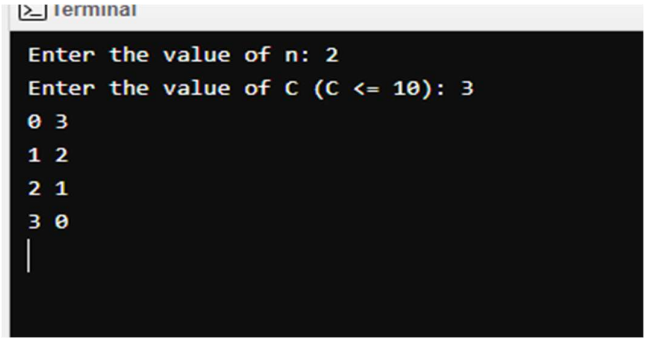
PRACTICAL 4

For any number n, write a program to list all the solutions of the equation x1 + x2 + x3 + ...+ xn = C, where C is a constant (C<=10) and x1, x2 ,x3,...,xn are nonnegative integers,using brute force strategy.

ANSWER:

```
#include <iostream>
#include <vector>
void generateSolutions(int n, int C, std::vector<int>&
solution, int currentIndex, int currentSum) {
 if (currentIndex == n) {
 if (currentSum == C) {
 for (int i = 0; i < n; ++i) {
 std::cout << solution[i] << " ";
 }
std::cout << std::endl;}}
 else {for (int i = 0; i <= C - currentSum; ++i) {
 solution[currentIndex] = i;
 generateSolutions(n, C, solution,
currentIndex + 1, currentSum + i);
 }
 }}
int main() {
 int n, C;
 std::cout << "Enter the value of n: ";
 std::cin >> n;
 std::cout << "Enter the value of C (C <= 10): ";
 std::cin >> C;
 std::vector<int> solution(n, 0);
 generateSolutions(n, C, solution, 0, 0);
 return 0;}
```

OUTPUT:



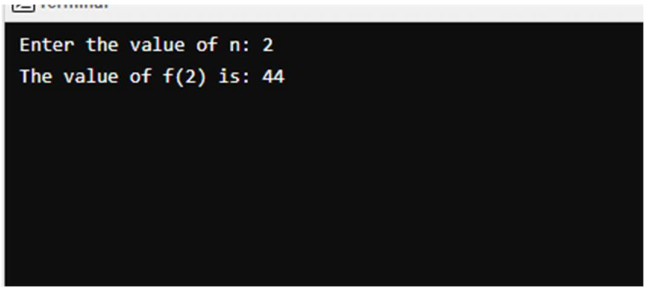
PRACTICAL 5

Write a Program to evaluate a polynomial function. (For example store f(x) = 4n2 + 2n + 9 in an array and for a given value of n, say n = 5, compute the value of f(n)).

ANSWER:

```
#include <iostream>
int main() {
 int coefficients[] = {4, 2, 9};
 int n;
 std::cout << "Enter the value of n: ";
 std::cin >> n;
 int result = 0;
 int powerOfN = 1;
 for (int i = 0; i < sizeof(coefficients) /
sizeof(coefficients[0]); i++) {
 result += coefficients[i] * powerOfN;
 powerOfN *= n;
 }
 std::cout << "The value of f(" << n << ") is: " <<
result << std::endl;
 return 0;
}
```

OUTPUT:



PRACTICAL 6

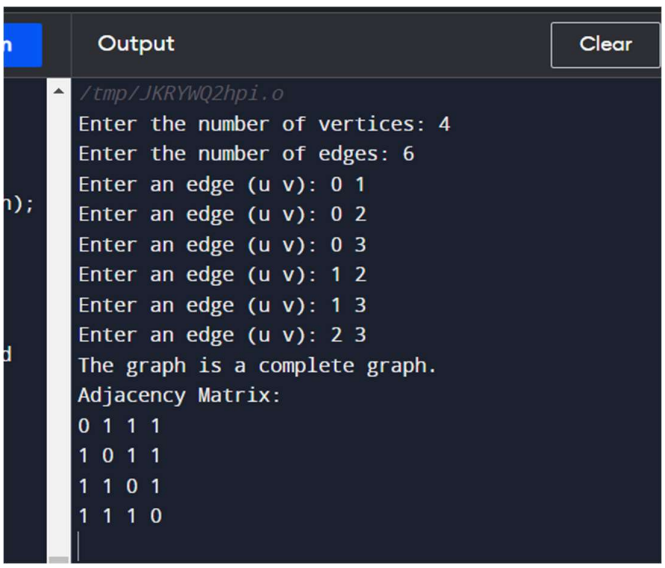
Write a Program to check if a given graph is a complete graph. Represent the graph using the Adjacency Matrix representation.

ANSWER:

```
#include <iostream>
#include <vector>
bool isCompleteGraph(const std::vector<std::vector<int>>
&graph)
{
 int n = graph.size();
 for (int i = 0; i < n; ++i)
 {
 for (int j = i + 1; j < n; ++j)
 {
 if (graph[i][j] == 0 || graph[j][i] == 0)
 {
 return false;
 }
 }
 }
 return true;
}
std::vector<std::vector<int>>
constructAdjacencyMatrix(int n, const
std::vector<std::pair<int, int>> &edges)
{
 std::vector<std::vector<int>> adjMatrix(n,
std::vector<int>(n, 0));
 for (const auto &edge : edges)
 {
 int u = edge.first;
 int v = edge.second;
 adjMatrix[u][v] = 1;
 adjMatrix[v][u] = 1;
 }
 return adjMatrix;
}
int main()
{
 int n, edgesCount;
 std::cout << "Enter the number of vertices: ";
```

```
 std::cin >> n;
 std::vector<std::pair<int, int>> edges;
 std::cout << "Enter the number of edges: ";
 std::cin >> edgesCount;
 for (int i = 0; i < edgesCount; ++i)
 {
 int u, v;
 std::cout << "Enter an edge (u v): ";
 std::cin >> u >> v;
 edges.emplace_back(u, v);
 }
 std::vector<std::vector<int>> graph =
constructAdjacencyMatrix(n, edges);
 bool isComplete = isCompleteGraph(graph);
 if (isComplete)
 {
 std::cout << "The graph is a complete graph." <<std::endl;
 std::cout << "Adjacency Matrix:" << std::endl;
 for (int i = 0; i < n; ++i)
 {
 for (int j = 0; j < n; ++j)
 {
 std::cout << graph[i][j] << " ";
 }
 std::cout << std::endl;
 }}else{std::cout << "The graph is not a complete graph." << std::endl;}
 return 0;
}
```

OUTPUT:



PRACTICAL 7

Write a Program to check if a given graph is a complete graph. Represent the graph using the Adjacency List representation.

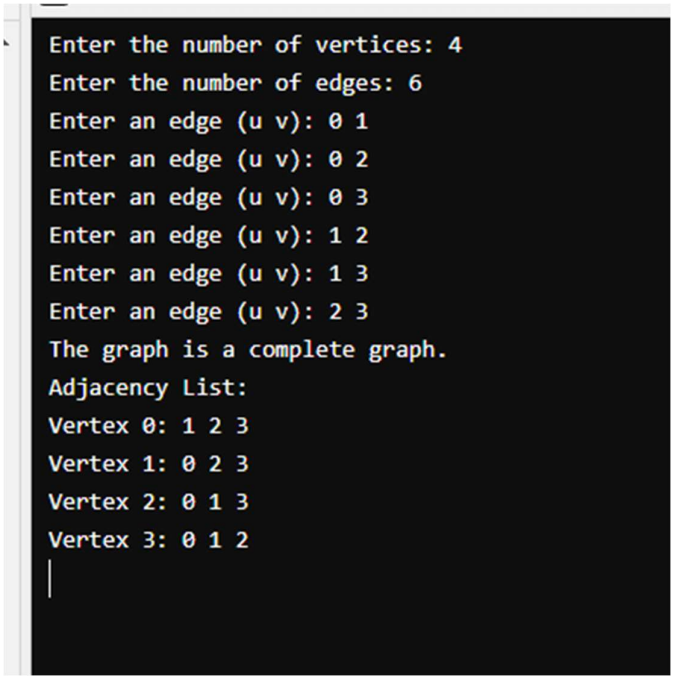
ANSWER:

```
#include <iostream>
#include <vector>
#include <list>
bool isCompleteGraph(const std::vector<std::list<int>>&
graph) {
 int n = graph.size();


 for (int i = 0; i < n; ++i) {
 for (int j = i + 1; j < n; ++j) {
 bool hasEdge = false;


 for (const auto& v : graph[i]) {
 if (v == j) {
 hasEdge = true;
 break;
 }
 }

 if (!hasEdge) {
 return false;
 }
 }
 }

 return true;
}
std::vector<std::list<int>> constructAdjacencyList(int
n, const std::vector<std::pair<int, int>>& edges) {
 std::vector<std::list<int>> adjList(n);


 for (const auto& edge : edges) {
 int u = edge.first;
 int v = edge.second;
```

```
 adjList[u].push_back(v);
 adjList[v].push_back(u);
 }

 return adjList;
}
int main() {
 int n, edgesCount;

 std::cout << "Enter the number of vertices: ";
 std::cin >> n;

 std::vector<std::pair<int, int>> edges;

 std::cout << "Enter the number of edges: ";
 std::cin >> edgesCount;


 for (int i = 0; i < edgesCount; ++i) {
 int u, v;
 std::cout << "Enter an edge (u v): ";
 std::cin >> u >> v;
 edges.emplace_back(u, v);
 }

 std::vector<std::list<int>> graph =
constructAdjacencyList(n, edges);

 bool isComplete = isCompleteGraph(graph);

 if (isComplete) {
 std::cout << "The graph is a complete graph." <<
std::endl;
 std::cout << "Adjacency List:" << std::endl;

 for (int i = 0; i < n; ++i) {
 std::cout << "Vertex " << i << ": ";
 for (const auto& v : graph[i]) {
 std::cout << v << " ";
 }
 std::cout << std::endl;
 }
 } else {
 std::cout << "The graph is not a complete graph." << std::endl;
```

```
}

return 0;
}
```

OUTPUT:



PRACTICAL 8

Write a Program to accept a directed graph G and compute the in-degree and out-degree of each vertex.

ANSWER:

```
#include <iostream>
#include <vector>
void computeDegrees(const std::vector<std::vector<int>>&
graph, std::vector<int>& inDegree, std::vector<int>&
outDegree) {
 int n = graph.size();


 inDegree.resize(n, 0);
 outDegree.resize(n, 0);


 for (int i = 0; i < n; ++i) {
 for (int j = 0; j < n; ++j) {
 if (graph[i][j] == 1) {

 outDegree[i]++;


 inDegree[j]++;
 }
 }
 }
}
int main() {
 int n;

 std::cout << "Enter the number of vertices: ";
 std::cin >> n;

 std::vector<std::vector<int>> graph(n,
std::vector<int>(n, 0));

 std::cout << "Enter the adjacency matrix of the directed graph:" << std::endl;
```

```
 for (int i = 0; i < n; ++i) {
 for (int j = 0; j < n; ++j) {
 std::cin >> graph[i][j];
 }
 }

 std::vector<int> inDegree, outDegree;

 computeDegrees(graph, inDegree, outDegree);

 std::cout << "Vertex\tIn-Degree\tOut-Degree" <<
std::endl;

 for (int i = 0; i < n; ++i) {
 std::cout << i << "\t" << inDegree[i] << "\t\t"
<< outDegree[i] << std::endl;
 }

 return 0;
}
```

OUTPUT: