Discrete Project

**Name: Danish $ Talha**

**Roll no: 21F-9229 21F-9134**

**Question 1: -**

//c++ program for the above given truth table's boolean expression

#include<iostream>

using namespace std;

//function that evaluates the given expression

int evaluate1(int p, int q) {

bool f;

f = !(p || q);

//if the expression's result is true then return 1 else return 0

if (f) {

return 1;

}

return 0;

}

//function that evaluates the given expression

int evaluate2(int p, int q) {

bool f;

f = !p && !q;

//if the expression's result is true then return 1 else return 0

if (f) {

return 1;

}

return 0;

}

//function that evaluates the given expression

int evaluate3(int p, int q, int r) {

bool f;

//if the expression's result is true then return 1 else return 0

f = (p || q) && !r;

if (f) {

return 1;

}

return 0;

}

//function that evaluates the given expression

int evaluate4(int p, int q, int r) {

bool f;

//if the expression's result is true then return 1 else return 0

f = !(p || (!q && r));

if (f) {

return 1;

}

return 0;

}

int main()

{

int p, q, r;

char ch;

//do while loop to make program runnable more n more if user want

do {

//taking values of p,q and r from user

cout << "\nEnter the value of 'p': ";

cin >> p;

cout << "\nEnter the value of 'q': ";

cin >> q;

cout << "\nEnter the value of 'r': ";

cin >> r;

//if p,q and r have value 0 or 1

//then we print the truth table

if ((p == 1 || p == 0) && (q == 1 || q == 0) && (r == 1 || r == 0)) {

int r1, r2, r3, r4;

r1 = evaluate1(p, q);

r2 = evaluate2(p, q);

r3 = evaluate3(p, q, r);

r4 = evaluate4(p, q, r);

cout << "\np\tq\tr\t!(p || q)\t!p && !q\t(p || q) && !r\t!(p || (!q && r))";

cout << "\n\n" << p << "\t" << q << "\t" << r << "\t " << r1 << "\t\t " << r2 << "\t\t " << r3 << "\t\t " << r4;

//user input if user want to run one more time

cout << "\nEnter Y/y for run the program again: ";

cin >> ch;

}

//else printing invalid and exit the program

else {

cout << "\nInvalid Input!!\nProgram Terminated";

exit(0);

}

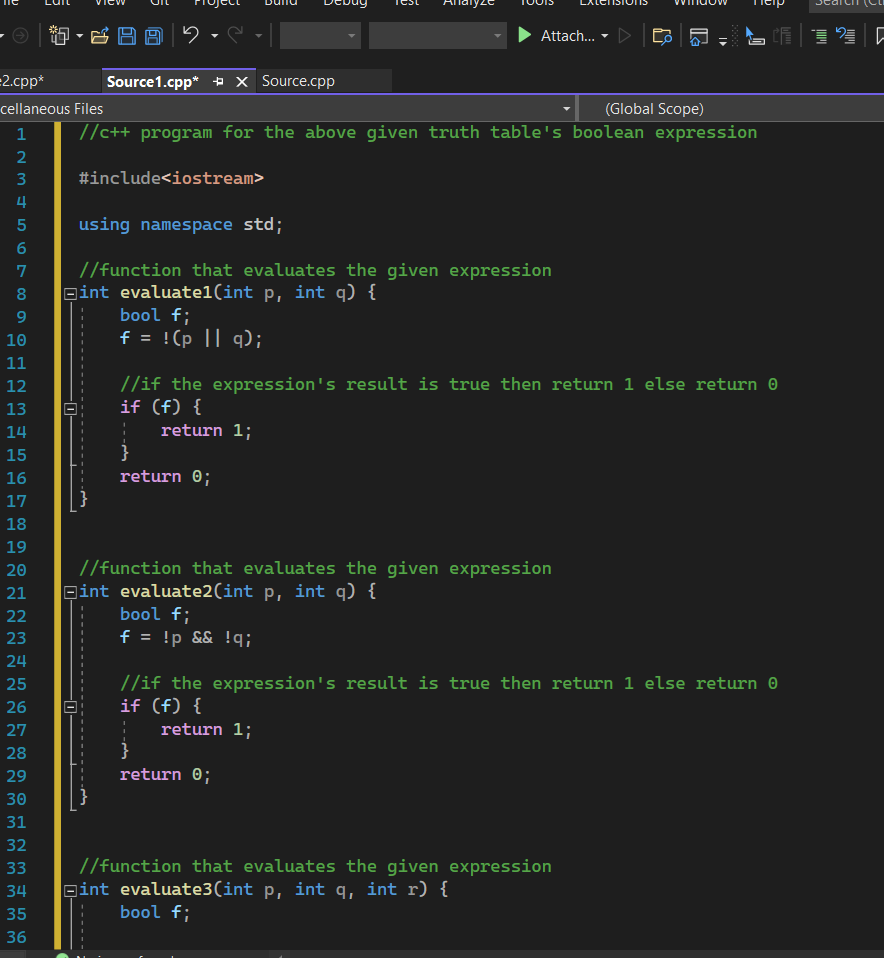
} while (ch == 'Y' || ch == 'y');

return 0;

}

**Working code:**

* Some screen shots

****

**Text

Description automatically generated**

**Question 2: -**

#include<iostream>

#include<iomanip>

using namespace std;

//ADT of node

struct node

{

int data;

node\* next;

};

//ADT of Linked List

class LL

{

public:

node\* head = nullptr;

node\* curr = nullptr;

//Member functions

LL();

bool isempty();

void insert\_end(int);

};

//Constructor

LL::LL()

{

head = nullptr;

curr = nullptr;

}

//CHeck if list is empty or not

bool LL::isempty()

{

if (head == NULL)

{

return true;

}

else

{

return false;

}

}

//Inserting new node at end

void LL::insert\_end(int value)

{

node\* temp = new node;

temp->data = value;

temp->next = NULL;

if (isempty() == true)

{

head = curr = temp;

}

else

{

curr->next = temp;

curr = curr->next;

}

}

//Graph ADT

class GraphADT

{

private:

int\*\* adjMatrix;

int\*\* PrimsMSTMatrix;

LL\* adjList;

LL\* PrimsList;

public:

int v;

char lastchar;

GraphADT(int);

void AddEdge(int src, int dest, int weight);

void CreateMST\_Prims();

void PrintMST\_Prims();

//int getmin(int\*\*, int&, int&);

void DisplayMatrix();

void DisplayList();

};

//Constructor to initialize matrix and list

GraphADT::GraphADT(int x)

{

v = x;

cout << "Number of vertexes : " << v << endl;

adjList = new LL[v];

PrimsList = new LL[v];

for (int i = 0; i < v; ++i)

{

adjList[i].head = NULL;

adjList[i].curr = adjList[i].head;

PrimsList[i].head = NULL;

PrimsList[i].curr = PrimsList[i].head;

}

adjMatrix = new int\* [v];

PrimsMSTMatrix = new int\* [v];

for (int i = 0; i < v; ++i)

{

adjMatrix[i] = new int[v];

PrimsMSTMatrix[i] = new int[v];

}

for (int i = 0; i < v; ++i)

{

for (int j = 0; j < v; ++j)

{

adjMatrix[i][j] = 0;

PrimsMSTMatrix[i][j] = 0;

}

}

lastchar = char(v + 64);

cout << "Available vertices : ";

for (int i = 0; i < v; ++i)

{

cout << " " << char(i + 65);

}

cout << endl;

}

void GraphADT::AddEdge(int src, int dest, int weight)

{

if (!(src >= 'A' && src <= lastchar))

{

cout << "Entered source vertex is invalid!\nDosent exist\n";

}

else if (!(dest >= 'A' && dest <= lastchar))

{

cout << "Entered Destination vertex is invalid!\nDosent exist\n";

}

src -= 65;

dest -= 65;

if (adjMatrix[src][dest] != 0)

{

cout << "Edge already exist!\n";

}

else

{

adjList[src].insert\_end(dest);

adjList[dest].insert\_end(src);

adjMatrix[src][dest] = weight;

adjMatrix[dest][src] = weight;

}

}

//Function to display Adjacency matrix

void GraphADT::DisplayMatrix()

{

cout << "\n";

cout << setfill('-');

cout << setw(44) << "Adjacency matrix" << setw(32);

cout << "\nV";

for (int i = 0; i < v; ++i)

{

cout << '\t' << char(i + 65);

}

cout << "\n |";

cout << "--------------------------------------------------------------------- ";

cout << endl;

for (int i = 0; i < v; ++i)

{

cout << char(i + 65) << " |";

for (int j = 0; j < v; ++j)

{

cout << "\t" << adjMatrix[i][j];

}

if (i != v - 1)

{

cout << "\n | \n";

}

}

cout << endl;

cout << setfill('-');

cout << setw(106);

}

//function to display the Adjacency list

void GraphADT::DisplayList()

{

cout << "\n\nDisplaying Adjacency List : \n\n";

for (int i = 0; i < v; ++i)

{

cout << "Vertex " << char(i + 65) << " :";

node\* curr = adjList[i].head;

while (curr != NULL)

{

cout << " -> " << char(curr->data + 65);

curr = curr->next;

}

cout << endl;

}

}

//Function to make MST by Prims Algorithm

void GraphADT::CreateMST\_Prims()

{

int\*\* temp;

temp = new int\* [v];

for (int i = 0; i < v; ++i)

{

temp[i] = new int[v];

}

for (int i = 0; i < v; ++i)

{

for (int j = 0; j < v; ++j)

{

temp[i][j] = adjMatrix[i][j];

}

}

int\* Vertex = new int[v];

for (int i = 0; i < v; ++i)

{

Vertex[i] = 0;

}

int edge\_count = 0;

Vertex[0] = 1;

cout << endl;

while (edge\_count < (v - 1))

{

int min = 20000;

int x = 0;

int y = 0;

for (int i = 0; i < v; i++)

{

if (Vertex[i] == 1)

{

for (int j = 0; j < v; j++)

{

if (!(Vertex[j] == 1) && temp[i][j])

{

if (min > temp[i][j])

{

min = temp[i][j];

x = i;

y = j;

}

}

}

}

}

PrimsMSTMatrix[x][y] = temp[x][y];

PrimsMSTMatrix[y][x] = temp[y][x];

PrimsList[x].insert\_end(y);

PrimsList[y].insert\_end(x);

edge\_count++;

temp[x][y] = 0;

Vertex[y] = 1;

}

}

//Function to display list and adj matrix of MST created by Prims Algorithm

void GraphADT::PrintMST\_Prims()

{

cout << "\n";

cout << "\n\nDisplaying Adjacency List : \n\n";

for (int i = 0; i < v; ++i)

{

cout << "Vertex " << char(i + 65) << " :";

node\* curr = PrimsList[i].head;

while (curr != NULL)

{

cout << " -> " << char(curr->data + 65);

curr = curr->next;

}

cout << endl;

}

cout << "\n";

cout << setfill('-');

cout << setw(44) << "Adjacency matrix" << setw(32);

cout << "\nV";

for (int i = 0; i < v; ++i)

{

cout << '\t' << char(i + 65);

}

cout << "\n |";

cout << "--------------------------------------------------------------------- ";

cout << endl;

for (int i = 0; i < v; ++i)

{

cout << char(i + 65) << " |";

for (int j = 0; j < v; ++j)

{

cout << "\t" << PrimsMSTMatrix[i][j];

}

if (i != v - 1)

{

cout << "\n | \n";

}

}

cout << endl;

cout << setfill('-');

cout << setw(106);

}

//Main program

int main()

{

GraphADT obj(9);

cout << "The graph is : \n";

obj.AddEdge('A', 'B', 9);

obj.AddEdge('A', 'C', 4);

obj.AddEdge('B', 'E', 7);

obj.AddEdge('B', 'D', 1);

obj.AddEdge('B', 'C', 2);

obj.AddEdge('C', 'D', 4);

obj.AddEdge('C', 'F', 3);

obj.AddEdge('D', 'E', 2);

obj.AddEdge('D', 'F', 5);

obj.AddEdge('E', 'F', 6);

obj.AddEdge('E', 'G', 3);

obj.AddEdge('F', 'G', 8);

obj.AddEdge('F', 'H', 5);

obj.AddEdge('G', 'H', 1);

obj.AddEdge('G', 'I', 3);

obj.AddEdge('H', 'I', 2);

obj.DisplayMatrix();

obj.DisplayList();

cout << "Now applying prims algorithm : \n";

obj.CreateMST\_Prims();

cout << "Displaying the minimum spanning tree : \n";

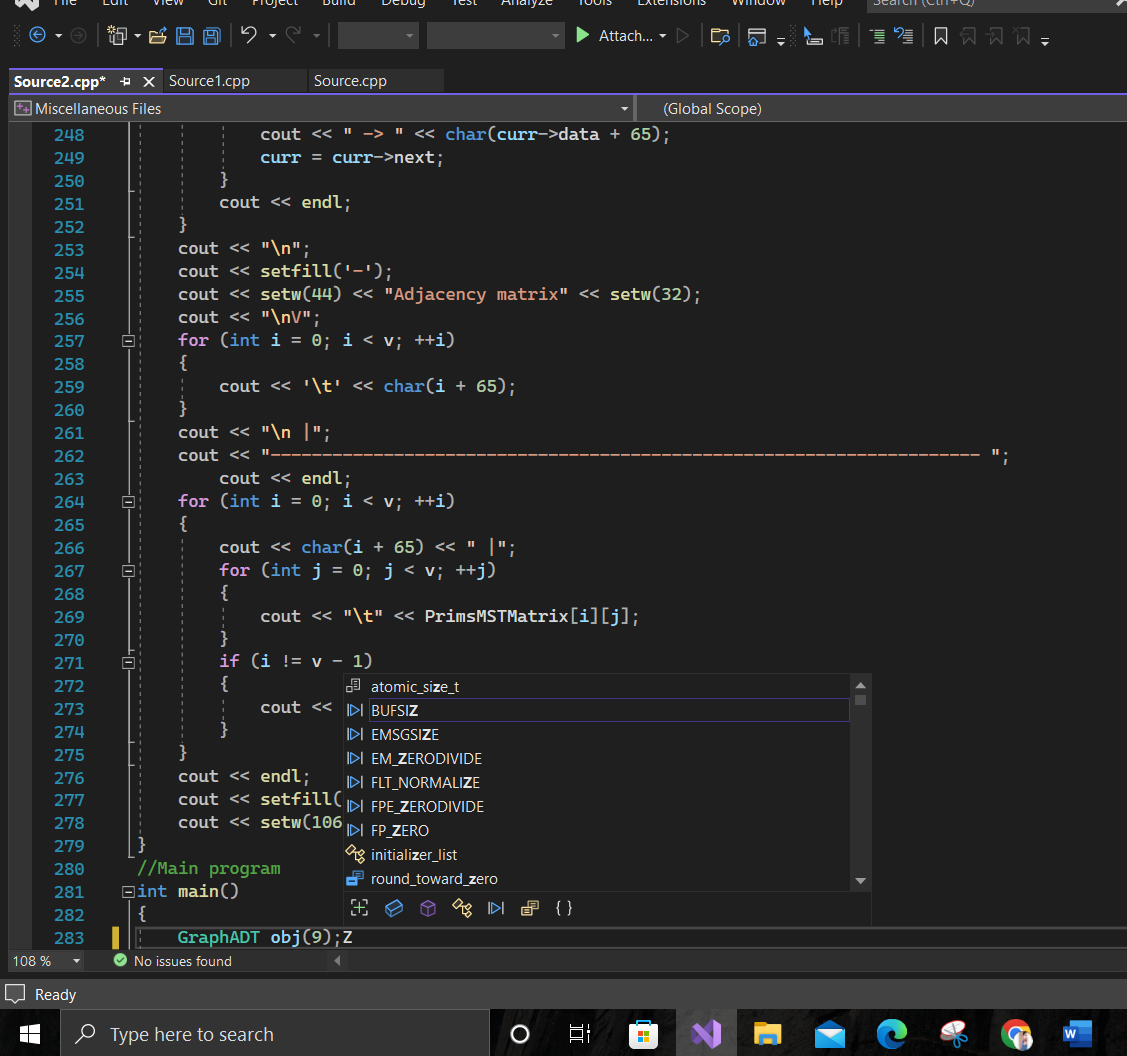
obj.PrintMST\_Prims();

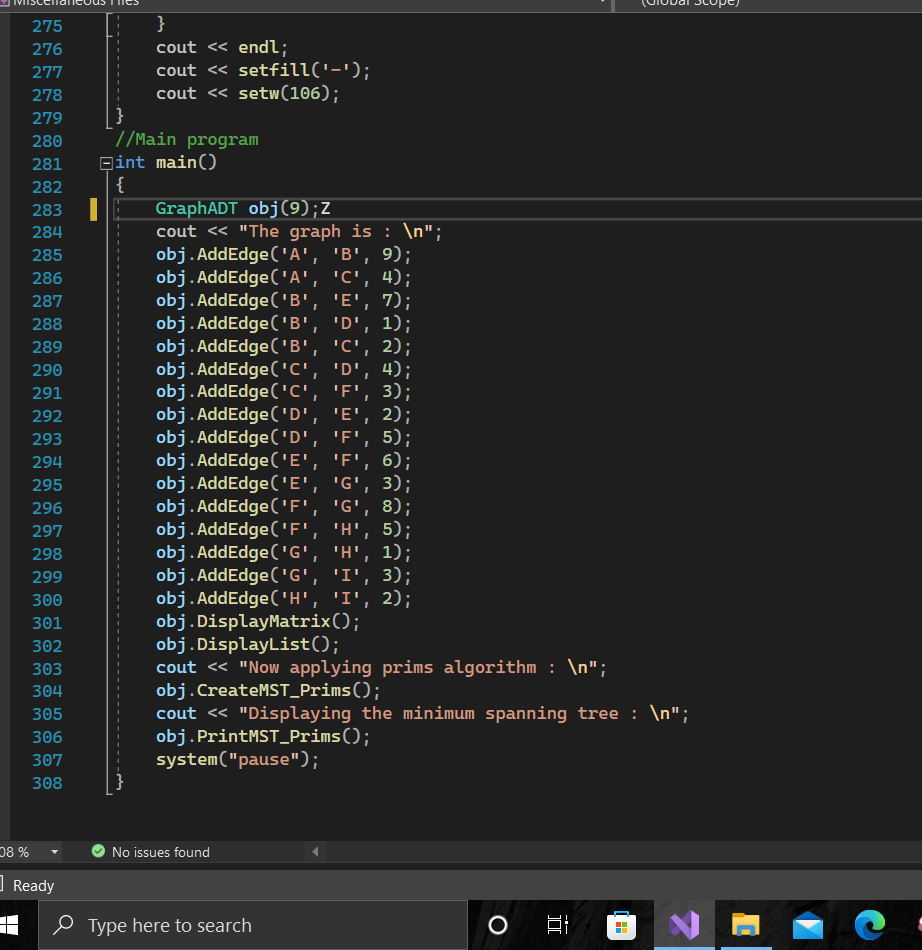
system("pause");

}

**Working code:**

* Only some screen shots





**Question 3: -**

**Logic:** Indeed, logic **plays an important role in areas of Computer Science** as disparate as artificial intelligence (automated reasoning), architecture (logic gates), software engineering (specification and verification), programming languages (semantics, logic programming), databases (relational algebra and SQL), algorithms.

Computational logic is **the use of computers to establish facts in a logical formalism**. Originating in nineteenth century attempts to understand the nature of mathematical reasoning, the subject now comprises a wide variety of formalisms, techniques, and technologies.

**Set:** In computer science, a set is an abstract data type that will **store an unordered collection of unique values**. In many programming languages, sets are implemented as built-in data structures, like arrays or dictionaries.

**Daily Life Examples of Sets**

* In Kitchen. Kitchen is the most relevant example of sets. ...
* School Bags. School bags of children is also an example. ...
* Shopping Malls. When we go shopping in a mall, we all have noticed that there are separate portions for each kind of things. ...
* Universe.
* Playlist.
* Rules.
* Representative House.

**Relations:** A relation in computer science is **a connection from a row of data to a column or type of data**. As such, relations result in tables. Relational databases, therefore, present data in tables. Relations are an important concept in set theory and its operations. Therefore, they play an important role in other concepts like functional analysis. The applications are broad-ranging and **sets the foundations for many other fields in set theory**.

**Example of a relation in real life:**

**Marriage** is one good example of relation and function on condition that its a faithful relationship.

**Probability:** Probability refers to **how likely an event is to occur**. Probability is used in all types of areas in real life including weather forecasting, sports betting, investing, and more.

**Applications Of Probability in Real Life:**

* Forecasting the weather.
* Sports outcomes.
* Card games and other games of chance.
* Insurance.
* Traffic signals.
* Medical diagnosis.
* Election results.
* Lottery probability.

**Minimal spanning trees:** Minimum spanning trees have direct applications in the design of networks, including computer networks, telecommunications networks, transportation networks, water supply networks, and electrical grids. A spanning tree is **a subset of Graph G, which has all the vertices covered with minimum possible number of edges**. Hence, a spanning tree does not have cycles and it cannot be disconnected. By this definition, we can draw a conclusion that every connected and undirected Graph G has at least one spanning tree.

**real life example of spanning trees**:

One example would be **a telecommunications company laying cable to a new neighborhood**. If it is constrained to bury the cable only along certain paths (e.g. along roads), then there would be a graph representing which points are connected by those paths.

**Graphs:** A graph is a non-linear data structure, which consists of vertices(or nodes) connected by edges(or arcs) where edges may be directed or undirected. In Computer science graphs are used **to represent the flow of computation.** Graphs come in many different flavors, many of which have found uses in computer programs. Some flavors are: **Simple graph**. **Undirected or directed graphs**.

**Helps to define the flow of computation of software programs**. Used in Google maps for building transportation systems. In google maps, the intersection of two or more roads represents the node while the road connecting two nodes represents an edge.

A simple graph is **a graph that does not contain more than one edge between the pair of vertices**. **A tree is a connected, undirected graph with no cycles**. Recall that a cycle is a path that starts and ends at the same node. In a tree, there are no cycles, which means that there is only one possible path between any two nodes. Searching in graphs **provides an appropriate level of abstraction within which to study simple problem solving independent of a particular domain**. A (directed) graph consists of a set of nodes and a set of directed arcs between nodes. The idea is to find a path along these arcs from a start node to a goal node.