**Task 1-A**

**Code**

#include<iostream>

//#include<string>

//#include<fstream>

//#include<vector>

//using namespace std;

//

//vector<string> seperateString(vector<char>, string);

//vector<char> getPremisis(string);

//int searchResult(vector<string>, string);

//int searchResult(vector<char>, char);

//int searchResult(vector<int>, int);

//vector<vector<bool>> setTruthTable(vector<char>, vector<string>);

//void displayChar(vector<char>);

//void displayString(vector<string>);

//void displayBool(vector<char>, vector<vector<bool>>, vector<string> premisis);

//bool checkTruthTableEquivalance(vector<char>, vector<vector<bool>>, vector<vector<bool>>);

//

//fstream write("Q1-A.csv", ios::out);

//

//int main()

//{

// // "(((p^q)->r)<->(q->r))"

// // "(((p->q)^(q->r))->(p->r))"

//

// // "(((~q)->(~p))V((~s)->(~r)))"

// // "(((p->q)^(r->s))^(pVr))"

//

// // "(((p^q)->r)<->(p->(q->r)))"

// // "(((p->q)^(q->r))->(p->r))"

//

// string rawStringLHS;

// string rawStringRHS;

// cout << "\t\t\*\*\* LOGICALLY EQUIVALENCE CHECKER \*\*\*\n";

// cout << "\n(1) 'V' for AND\n(2) '^' for OR\n(3) '~' for NOT\n(4) '->' for IMPLICATION\n(5) '<->' for DOUBLE IMPLICATION'\n\n\*\*\* BRACKETS ARE ABSOLUTELLY NECCESSARY \*\*\*\n";

// cout << "Enter LHS premises : ";

// cin >> rawStringLHS;

// cout << "Enter RHS premises : ";

// cin >> rawStringRHS;

// vector<char> premisis = getPremisis(rawStringLHS + rawStringRHS);

// vector<string> premisisLHS = seperateString(premisis, rawStringLHS);

// vector<string> premisisRHS = seperateString(premisis, rawStringRHS);

//

// vector<vector<bool>> truthTableLHS = setTruthTable(premisis, premisisLHS);

// displayString(premisisLHS);

// displayBool(premisis, truthTableLHS, premisisLHS);

//

// cout << endl << endl;

//

// vector<vector<bool>> truthTableRHS = setTruthTable(premisis, premisisRHS);

// displayString(premisisRHS);

// displayBool(premisis, truthTableRHS, premisisRHS);

//

// bool checkEquivalance = checkTruthTableEquivalance(premisis, truthTableLHS, truthTableRHS);

//

// if (checkEquivalance)

// {

// cout << "\nTruth Table Is Equivalant\n";

// write << "\nTruth Table Is Equivalant\n";

// }

// else

// {

// cout << "\nTruth Table Is Not Equivalant\n";

// write << "\nTruth Table Is Not Equivalant\n";

// }

//

// return 0;

//}

//

//bool Implication(bool x, bool y)

//{

// if (y == true)

// {

// return true;

// }

// else if (x == false and y == false)

// {

// return true;

// }

// else

// {

// return false;

// }

//}

//

//bool And(bool x, bool y)

//{

// if (x == true and y == true)

// {

// return true;

// }

// else

// {

// return false;

// }

//}

//

//bool Or(bool x, bool y)

//{

// if (x == true or y == true)

// {

// return true;

// }

// else

// {

// return false;

// }

//}

//

//bool Negation(bool x)

//{

// if (x == true)

// {

// return false;

// }

// else

// {

// return true;

// }

//}

//

//bool BiConditional(bool x, bool y)

//{

// if (x == true and y == true)

// {

// return true;

// }

// else if (x == false and y == false)

// {

// return true;

// }

// else

// {

// return false;

// }

//}

//

//vector<vector<bool>> setTruthTable(vector<char>premisis, vector<string> hs)

//{

// int rows = pow(2, premisis.size());

// int column = hs.size();

// vector<vector<bool>> truthTable(rows);

//

// for (int i = 0; i < rows; i++)

// {

// truthTable[i].resize(column);

//

// for (int j = 0; j < column; j++)

// {

// truthTable[i][j] = false;

// }

// }

//

// int num = 2;

// int count = 0;

// bool check;

//

// for (int i = 0; i < premisis.size(); i++)

// {

// count = rows / num;

// check = true;

//

// for (int j = 0; j < rows; j++)

// {

// if (count > 0)

// {

// truthTable[j][i] = check;

// count--;

// }

// else if (count <= 0 and check == true)

// {

// count = (rows / num) - 1;

// check = false;

//

// truthTable[j][i] = check;

// }

// else if (count <= 0 and check == false)

// {

// count = (rows / num) - 1;

// check = true;

//

// truthTable[j][i] = check;

// }

// }

//

// num \*= 2;

// }

//

// int curr = -1;

//

// for (int i = premisis.size(); i < truthTable[0].size(); i++)

// {

// curr = i;

// string s1 = "", s2 = hs[curr];

//

// int index1 = -1, index2 = -1;

//

// for (int sIndex = curr - 1; sIndex >= 0; sIndex--)

// {

// s1 = hs[sIndex];

//

// int found1 = s2.find(s1);

// if (found1 != string::npos)

// {

// index1 = sIndex;

// string s = "";

//

// if (("(~" + s1 + ')') == s2)

// {

// for (int j = 0; j < rows; j++)

// {

// truthTable[j][i] = Negation(truthTable[j][index1]);

// }

// }

//

// if ("(" + s1 + ")" == s2)

// {

// for (int j = 0; j < rows; j++)

// {

// truthTable[j][i] = truthTable[j][index1];

// }

// }

//

// for (int ssIndex = curr - 1; ssIndex >= 0; ssIndex--)

// {

// string s3 = hs[ssIndex];

// int found2 = s2.find(s3);

// if (found2 != string::npos and found1 != found2)

// {

// index2 = ssIndex;

//

// if (('(' + s1 + "^" + s3 + ')') == s2)

// {

// for (int j = 0; j < rows; j++)

// {

// truthTable[j][i] = And(truthTable[j][index1], truthTable[j][index2]);

// }

// }

// else if (('(' + s1 + "~" + s3 + ')') == s2)

// {

// for (int j = 0; j < rows; j++)

// {

// truthTable[j][i] = And(truthTable[j][index1], truthTable[j][index2]);

// }

// }

// else if (('(' + s1 + "V" + s3 + ')') == s2)

// {

// for (int j = 0; j < rows; j++)

// {

// truthTable[j][i] = Or(truthTable[j][index1], truthTable[j][index2]);

// }

// }

// else if (('(' + s1 + "->" + s3 + ')') == s2)

// {

// for (int j = 0; j < rows; j++)

// {

// truthTable[j][i] = Implication(truthTable[j][index1], truthTable[j][index2]);

// }

// }

// else if (('(' + s1 + "<->" + s3 + ')') == s2)

// {

// for (int j = 0; j < rows; j++)

// {

// truthTable[j][i] = BiConditional(truthTable[j][index1], truthTable[j][index2]);

// }

// }

//

// else if (('(' + s3 + "^" + s1 + ')') == s2)

// {

// for (int j = 0; j < rows; j++)

// {

// truthTable[j][i] = And(truthTable[j][index2], truthTable[j][index1]);

// }

// }

// else if (('(' + s3 + "~" + s1 + ')') == s2)

// {

// for (int j = 0; j < rows; j++)

// {

// truthTable[j][i] = And(truthTable[j][index2], truthTable[j][index1]);

// }

// }

// else if (('(' + s3 + "V" + s1 + ')') == s2)

// {

// for (int j = 0; j < rows; j++)

// {

// truthTable[j][i] = Or(truthTable[j][index2], truthTable[j][index1]);

// }

// }

// else if (('(' + s3 + "->" + s1 + ')') == s2)

// {

// for (int j = 0; j < rows; j++)

// {

// truthTable[j][i] = Implication(truthTable[j][index2], truthTable[j][index1]);

// }

// }

// else if (('(' + s3 + "<->" + s1 + ')') == s2)

// {

// for (int j = 0; j < rows; j++)

// {

// truthTable[j][i] = BiConditional(truthTable[j][index2], truthTable[j][index1]);

// }

// }

// }

// }

// }

// }

// }

//

// return truthTable;

//}

//

//void displayBool(vector<vector<bool>> truthTable)

//{

// for (int i = 0; i < truthTable.size(); i++)

// {

// cout << endl;

// for (int j = 0; j < truthTable[i].size(); j++)

// {

// cout << "|" << truthTable[i][j] << "|\t";

// }

// cout << endl;

// }

//}

//

//void displayBool(vector<char> prem, vector<vector<bool>> truthTable, vector<string> premisis)

//{

// int max = 0;

//

// for (int curr = 0; curr < premisis.size(); curr++)

// {

// if (premisis[curr].size() > max)

// {

// max = premisis[curr].size();

// }

// }

//

// for (int i = 0; i < truthTable.size(); i++)

// {

// cout << endl;

// write << endl;

// for (int j = 0; j < truthTable[i].size(); j++)

// {

// cout << "| " << truthTable[i][j] << " |";

// write << truthTable[i][j] << ",";

// }

// cout << endl;

// }

//

// write << endl;

//}

//

//vector<string> seperateString(vector<char> pre, string rawString)

//{

// vector<string> premisis;

// vector<int> brackOpenPosition;

// vector<int> brackClosePosition;

//

// for (int i = 0; i < pre.size(); i++)

// {

// string temp = "";

// temp += pre[i];

//

// premisis.push\_back(temp);

// }

//

// int brackOpenCount = 0, brackCloseCount = 0;

// char openBrack = '(', closeBrack = ')';

//

// for (int curr = 0; curr < rawString.length(); curr++)

// {

// if (rawString[curr] == openBrack)

// {

// brackOpenCount++;

// }

// else if (rawString[curr] == closeBrack)

// {

// brackCloseCount++;

// }

// }

//

// if (brackOpenCount != brackCloseCount)

// {

// cout << "Error :: Bracket Missing";

// exit(0);

// }

//

// int brackCount = brackOpenCount;

//

// for (int curr = 0; curr < rawString.length(); curr++)

// {

// if (rawString[curr] == closeBrack && (searchResult(brackClosePosition, curr) == -1))

// {

// brackClosePosition.push\_back(curr);

//

// for (int i = curr; i >= 0; i--)

// {

// if (rawString[i] == openBrack && (searchResult(brackOpenPosition, i) == -1))

// {

// brackOpenPosition.push\_back(i);

// break;

// }

// }

// }

// }

//

// string temp = "";

//

// for (int curr = 0; curr < brackCount; curr++)

// {

// temp = "";

//

// for (int i = brackOpenPosition[curr]; i <= brackClosePosition[curr]; i++)

// {

// temp += rawString[i];

// }

//

// premisis.push\_back(temp);

// }

//

// return premisis;

//}

//

//vector<char> getPremisis(string main)

//{

// vector<char> premisis;

//

// for (int i = 0; i < main.length(); i++)

// {

// for (int curr = 97; curr <= 122; curr++)

// {

// char temp1 = curr, temp2 = main[i];

//

// if (temp1 == temp2 && searchResult(premisis, temp2) == -1)

// {

// premisis.push\_back(temp2);

// }

// }

// }

//

// for (int i = 0; i < premisis.size() - 1; i++)

// {

// if (premisis[i] > premisis[i + 1])

// {

// char temp = premisis[i];

// premisis[i] = premisis[i + 1];

// premisis[i + 1] = temp;

// }

// }

//

// return premisis;

//}

//

//int searchResult(vector<char> arr, char k)

//{

// vector<char>::iterator it;

// it = find(arr.begin(), arr.end(), k);

//

// if (it != arr.end())

// {

// return (it - arr.begin());

// }

// else

// {

// return -1;

// }

//}

//

//int searchResult(vector<string> arr, string k)

//{

// vector<string>::iterator it;

// it = find(arr.begin(), arr.end(), k);

//

// if (it != arr.end())

// {

// return (it - arr.begin());

// }

// else

// {

// return -1;

// }

//}

//

//int searchResult(vector<int> arr, int k)

//{

// vector<int>::iterator it;

// it = find(arr.begin(), arr.end(), k);

//

// if (it != arr.end())

// {

// return (it - arr.begin());

// }

// else

// {

// return -1;

// }

//}

//

//void displayChar(vector<char> premisis)

//{

// for (int curr = 0; curr < premisis.size(); curr++)

// {

// cout << premisis[curr] << "\t";

// }

//}

//

//void displayString(vector<string> premisis)

//{

// int max = 0;

//

// for (int curr = 0; curr < premisis.size(); curr++)

// {

// if (premisis[curr].size() > max)

// {

// max = premisis[curr].size();

// }

// }

//

// cout << endl;

// write << endl;

//

// for (int curr = 0; curr < premisis.size(); curr++)

// {

// int size = max - premisis[curr].size();

// cout << "\t" << premisis[curr];

// write << premisis[curr] << ",";

// }

// write << endl;

// cout << endl;

//}

//

//bool checkTruthTableEquivalance(vector<char> prem, vector<vector<bool>>LHS, vector<vector<bool>>RHS)

//{

//

// for (int i = 0; i < LHS.size(); i++)

// {

// int indexLHS = LHS[i].size() - 1;

// int indexRHS = RHS[i].size() - 1;

//

// if (LHS[i][indexLHS] != RHS[i][indexRHS])

// {

// return false;

// }

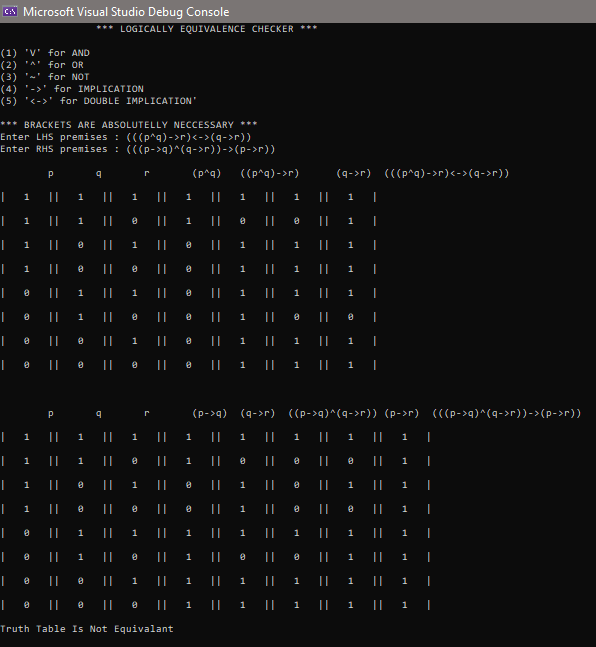
// }

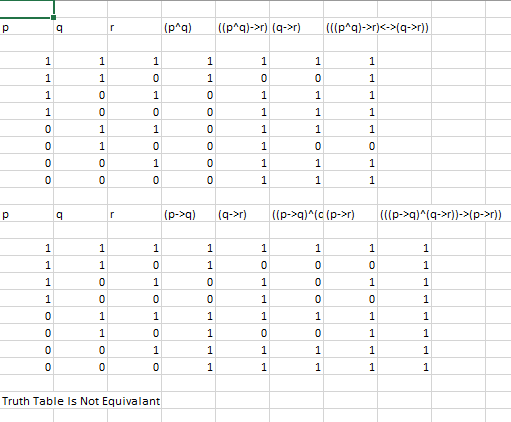
//

// return true;

//}

**Output**

****

****

**Task 1-B**

**Code**

//#include<iostream>

//#include<fstream>

//#include<vector>

//using namespace std;

//

//vector<string> seperateString(vector<char>, string);

//vector<char> getPremisis(string);

//int searchResult(vector<string>, string);

//int searchResult(vector<char>, char);

//int searchResult(vector<int>, int);

//vector<vector<bool>> setTruthTable(vector<char>, vector<string>);

//void displayChar(vector<char>);

//void displayString(vector<string>);

//void displayBool(vector<string>, vector<vector<bool>>);

//void display(vector<vector<bool>>);

//bool checkTruthTableEquivalance(vector<char>, vector<vector<bool>>, vector<vector<bool>>);

//vector<bool> getLastColumn(vector<string>, vector<vector<bool>>);

//bool checkValidation(vector<vector<bool>>);

//

//fstream write("Q1-B.csv", ios::out);

//

//int main()

//{

// vector<vector<bool>> main;

// vector<string> allPremisis;

// string rawString = "", prem = "";

// int i = 1;

// while (true)

// {

// cout << "\t\t\*\*\* ARGUMENT VALIDATOR \*\*\*\n";

// cout << "\n(1) 'V' for OR\n(2) '^' for AND\n(3) '~' for NOT\n(4) '->' for IMPLICATION\n(5) '<->' for DOUBLE IMPLICATION'\n\n\*\*\* BRACKETS ARE ABSOLUTELLY NECCESSARY \*\*\*\n";

// cout << "\nEnter Premisis No. " << i << "\n(\*PRESS 0 to enter the CONCLUSION\*)\nInput: ";

// cin >> rawString;

//

// if (rawString == "0")

// {

// break;

// }

//

// prem += rawString;

// allPremisis.push\_back(rawString);

// i++;

// }

//

// cout << "\nEnter Conclusion: ";

// cin >> rawString;

//

// prem += rawString;

// allPremisis.push\_back(rawString);

//

// vector<char> mainPremisis = getPremisis(prem);

//

// for (int i = 0; i < allPremisis.size(); i++)

// {

// vector<string> tempPremisis = seperateString(mainPremisis, allPremisis[i]);

// vector<vector<bool>> truthTable = setTruthTable(mainPremisis, tempPremisis);

// vector<bool> lastColumn = getLastColumn(tempPremisis, truthTable);

// main.push\_back(lastColumn);

// }

//

// cout << "\n\nMain Truth Table\n";

// displayString(allPremisis);

// display(main);

// bool validate = checkValidation(main);

//

// if (validate)

// {

// cout << "None\n";

// write << "None\n";

// }

//

// cout << (validate ? "\nArgument is Valid\n" : "\nArgument is Not Valid\n");

// write << (validate ? "\nArgument is Valid\n" : "\nArgument is Not Valid\n");

//

// return 0;

//}

//

//vector<bool> getLastColumn(vector<string> prem, vector<vector<bool>> truthTable)

//{

// vector<bool> lastColumn;

//

// cout << endl << endl;

// displayString(prem);

// displayBool(prem, truthTable);

//

// for (int i = 0; i < truthTable.size(); i++)

// {

// lastColumn.push\_back(truthTable[i][prem.size() - 1]);

// }

//

// return lastColumn;

//}

//

//bool Implication(bool x, bool y)

//{

// if (y == true)

// {

// return true;

// }

// else if (x == false and y == false)

// {

// return true;

// }

// else

// {

// return false;

// }

//}

//

//bool And(bool x, bool y)

//{

// if (x == true and y == true)

// {

// return true;

// }

// else

// {

// return false;

// }

//}

//

//bool Or(bool x, bool y)

//{

// if (x == true or y == true)

// {

// return true;

// }

// else

// {

// return false;

// }

//}

//

//bool Negation(bool x)

//{

// if (x == true)

// {

// return false;

// }

// else

// {

// return true;

// }

//}

//

//bool BiConditional(bool x, bool y)

//{

// if (x == true and y == true)

// {

// return true;

// }

// else if (x == false and y == false)

// {

// return true;

// }

// else

// {

// return false;

// }

//}

//

//vector<vector<bool>> setTruthTable(vector<char>premisis, vector<string> hs)

//{

// int rows = pow(2, premisis.size());

// int column = hs.size();

// vector<vector<bool>> truthTable(rows);

//

// for (int i = 0; i < rows; i++)

// {

// truthTable[i].resize(column);

//

// for (int j = 0; j < column; j++)

// {

// truthTable[i][j] = false;

// }

// }

//

// int num = 2;

// int count = 0;

// bool check;

//

// for (int i = 0; i < premisis.size(); i++)

// {

// count = rows / num;

// check = true;

//

// for (int j = 0; j < rows; j++)

// {

// if (count > 0)

// {

// truthTable[j][i] = check;

// count--;

// }

// else if (count <= 0 and check == true)

// {

// count = (rows / num) - 1;

// check = false;

//

// truthTable[j][i] = check;

// }

// else if (count <= 0 and check == false)

// {

// count = (rows / num) - 1;

// check = true;

//

// truthTable[j][i] = check;

// }

// }

//

// num \*= 2;

// }

//

// int curr = -1;

//

// for (int i = premisis.size(); i < truthTable[0].size(); i++)

// {

// curr = i;

// string s1 = "", s2 = hs[curr];

//

// int index1 = -1, index2 = -1;

//

// for (int sIndex = curr - 1; sIndex >= 0; sIndex--)

// {

// s1 = hs[sIndex];

//

// int found1 = s2.find(s1);

// if (found1 != string::npos)

// {

// index1 = sIndex;

// string s = "";

//

// if (("(~" + s1 + ')') == s2)

// {

// for (int j = 0; j < rows; j++)

// {

// truthTable[j][i] = Negation(truthTable[j][index1]);

// }

// }

//

// if ("(" + s1 + ")" == s2)

// {

// for (int j = 0; j < rows; j++)

// {

// truthTable[j][i] = truthTable[j][index1];

// }

// }

//

// for (int ssIndex = curr - 1; ssIndex >= 0; ssIndex--)

// {

// string s3 = hs[ssIndex];

// int found2 = s2.find(s3);

// if (found2 != string::npos and found1 != found2)

// {

// index2 = ssIndex;

//

// if (('(' + s1 + "^" + s3 + ')') == s2)

// {

// for (int j = 0; j < rows; j++)

// {

// truthTable[j][i] = And(truthTable[j][index1], truthTable[j][index2]);

// }

// }

// else if (('(' + s1 + "~" + s3 + ')') == s2)

// {

// for (int j = 0; j < rows; j++)

// {

// truthTable[j][i] = And(truthTable[j][index1], truthTable[j][index2]);

// }

// }

// else if (('(' + s1 + "V" + s3 + ')') == s2)

// {

// for (int j = 0; j < rows; j++)

// {

// truthTable[j][i] = Or(truthTable[j][index1], truthTable[j][index2]);

// }

// }

// else if (('(' + s1 + "->" + s3 + ')') == s2)

// {

// for (int j = 0; j < rows; j++)

// {

// truthTable[j][i] = Implication(truthTable[j][index1], truthTable[j][index2]);

// }

// }

// else if (('(' + s1 + "<->" + s3 + ')') == s2)

// {

// for (int j = 0; j < rows; j++)

// {

// truthTable[j][i] = BiConditional(truthTable[j][index1], truthTable[j][index2]);

// }

// }

//

// else if (('(' + s3 + "^" + s1 + ')') == s2)

// {

// for (int j = 0; j < rows; j++)

// {

// truthTable[j][i] = And(truthTable[j][index2], truthTable[j][index1]);

// }

// }

// else if (('(' + s3 + "~" + s1 + ')') == s2)

// {

// for (int j = 0; j < rows; j++)

// {

// truthTable[j][i] = And(truthTable[j][index2], truthTable[j][index1]);

// }

// }

// else if (('(' + s3 + "V" + s1 + ')') == s2)

// {

// for (int j = 0; j < rows; j++)

// {

// truthTable[j][i] = Or(truthTable[j][index2], truthTable[j][index1]);

// }

// }

// else if (('(' + s3 + "->" + s1 + ')') == s2)

// {

// for (int j = 0; j < rows; j++)

// {

// truthTable[j][i] = Implication(truthTable[j][index2], truthTable[j][index1]);

// }

// }

// else if (('(' + s3 + "<->" + s1 + ')') == s2)

// {

// for (int j = 0; j < rows; j++)

// {

// truthTable[j][i] = BiConditional(truthTable[j][index2], truthTable[j][index1]);

// }

// }

// }

// }

// }

// }

// }

//

// return truthTable;

//}

//

//bool checkValidation(vector<vector<bool>> truthTable)

//{

// bool fal = true, check = true;

// cout << "\n\nInvalid Arguments: \n";

//

// for (int i = 0; i < truthTable[0].size(); i++)

// {

// check = true;

//

// for (int j = 0; j < truthTable.size(); j++)

// {

// if (truthTable[j][i] == false && j != truthTable.size() - 1)

// {

// check = false;

// }

//

// if (check == true && j == truthTable.size() - 1 && truthTable[j][i] == false)

// {

// for (int curr = 0; curr < truthTable.size(); curr++)

// {

// cout << truthTable[curr][i] << "\t";

// }

// cout << endl;

// fal = false;

// check = false;

// }

// }

// }

//

// return (check && fal);

//}

//

//void displayBool(vector<string> prem, vector<vector<bool>> truthTable)

//{

// cout << endl;

// write << endl;

//

// for (int i = 0; i < truthTable.size(); i++)

// {

// //for (int j = 0; j < truthTable[0].size() - prem.size(); j++)

// for (int j = 0; j < truthTable[i].size(); j++)

// {

// cout << truthTable[i][j] << "\t";

// write << truthTable[i][j] << ",";

// }

// cout << endl;

// write << endl;

// }

//}

//

//void display(vector<vector<bool>> truthTable)

//{

// cout << endl;

// write << endl;

// for (int i = 0; i < truthTable[0].size(); i++)

// {

// for (int j = 0; j < truthTable.size(); j++)

// {

// cout << truthTable[j][i] << "\t";

// write << truthTable[j][i] << ",";

// }

// cout << endl;

// write << endl;

// }

//}

//

//vector<string> seperateString(vector<char> pre, string rawString)

//{

// vector<string> premisis;

// vector<int> brackOpenPosition;

// vector<int> brackClosePosition;

//

// for (int i = 0; i < pre.size(); i++)

// {

// string temp = "";

// temp += pre[i];

//

// premisis.push\_back(temp);

// }

//

// int brackOpenCount = 0, brackCloseCount = 0;

// char openBrack = '(', closeBrack = ')';

//

// for (int curr = 0; curr < rawString.length(); curr++)

// {

// if (rawString[curr] == openBrack)

// {

// brackOpenCount++;

// }

// else if (rawString[curr] == closeBrack)

// {

// brackCloseCount++;

// }

// }

//

// if (brackOpenCount != brackCloseCount)

// {

// cout << "Error :: Bracket Missing";

// exit(0);

// }

//

// int brackCount = brackOpenCount;

//

// for (int curr = 0; curr < rawString.length(); curr++)

// {

// if (rawString[curr] == closeBrack && (searchResult(brackClosePosition, curr) == -1))

// {

// brackClosePosition.push\_back(curr);

//

// for (int i = curr; i >= 0; i--)

// {

// if (rawString[i] == openBrack && (searchResult(brackOpenPosition, i) == -1))

// {

// brackOpenPosition.push\_back(i);

// break;

// }

// }

// }

// }

//

// string temp = "";

//

// for (int curr = 0; curr < brackCount; curr++)

// {

// temp = "";

//

// for (int i = brackOpenPosition[curr]; i <= brackClosePosition[curr]; i++)

// {

// temp += rawString[i];

// }

//

// premisis.push\_back(temp);

// }

//

// return premisis;

//}

//

//vector<char> getPremisis(string main)

//{

// vector<char> premisis;

//

// for (int i = 0; i < main.length(); i++)

// {

// for (int curr = 97; curr <= 122; curr++)

// {

// char temp1 = curr, temp2 = main[i];

//

// if (temp1 == temp2 && searchResult(premisis, temp2) == -1)

// {

// premisis.push\_back(temp2);

// }

// }

// }

//

// for (int i = 0; i < premisis.size() - 1; i++)

// {

// if (premisis[i] > premisis[i + 1])

// {

// char temp = premisis[i];

// premisis[i] = premisis[i + 1];

// premisis[i + 1] = temp;

// }

// }

//

// return premisis;

//}

//

//int searchResult(vector<char> arr, char k)

//{

// vector<char>::iterator it;

// it = find(arr.begin(), arr.end(), k);

//

// if (it != arr.end())

// {

// return (it - arr.begin());

// }

// else

// {

// return -1;

// }

//}

//

//int searchResult(vector<string> arr, string k)

//{

// vector<string>::iterator it;

// it = find(arr.begin(), arr.end(), k);

//

// if (it != arr.end())

// {

// return (it - arr.begin());

// }

// else

// {

// return -1;

// }

//}

//

//int searchResult(vector<int> arr, int k)

//{

// vector<int>::iterator it;

// it = find(arr.begin(), arr.end(), k);

//

// if (it != arr.end())

// {

// return (it - arr.begin());

// }

// else

// {

// return -1;

// }

//}

//

//void displayChar(vector<char> premisis)

//{

// for (int curr = 0; curr < premisis.size(); curr++)

// {

// cout << premisis[curr] << "\t";

// }

//}

//

//void displayString(vector<string> premisis)

//{

// cout << endl;

// write << endl;

//

// for (int curr = 0; curr < premisis.size(); curr++)

// {

// cout << premisis[curr] << "\t";

// write << premisis[curr] << ",";

// }

//

// write << endl;

// cout << endl;

//}

//

//bool checkTruthTableEquivalance(vector<char> prem, vector<vector<bool>>LHS, vector<vector<bool>>RHS)

//{

// int notSize = prem.size();

// int index = LHS.size();

//

// for (int i = 0; i < LHS.size(); i++)

// {

// if (LHS[index][i] != RHS[index][i])

// {

// return false;

// }

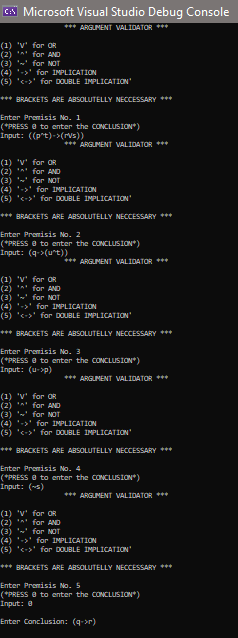
// }

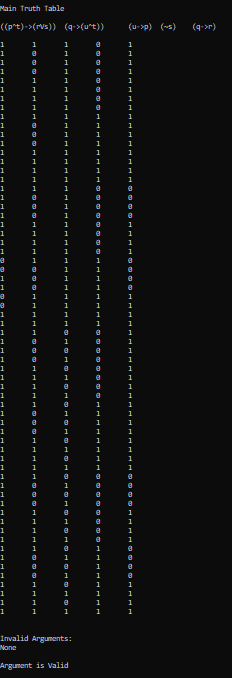
//

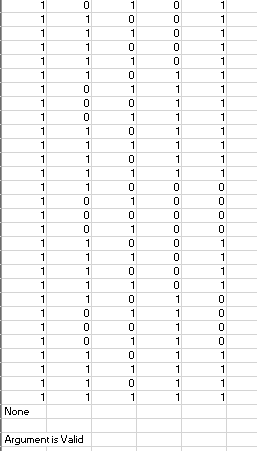
// return true;

//}

**Output**

****

****

****

**Task 2**

**Code**

//#include<iostream>

//#include<vector>

//using namespace std;

//

//void display(vector<int> causingVertices)

//{

// if (causingVertices.size() > 1)

// {

// cout << "\nVertices causing conflict: ";

// }

// else

// {

// cout << "Vertex causing conflict: ";

// }

//

// for (int j = 0; j < causingVertices.size(); j++)

// {

// cout << causingVertices[j];

//

// if (j < causingVertices.size() - 1)

// {

// cout << ", ";

// }

// }

// cout << endl;

//}

//

//vector<vector<int>> matrixToAdjList(vector<vector<int>> graph)

//{

// vector<vector<int>> adj\_list(graph.size());

//

// for (int i = 0; i < graph.size(); i++)

// {

// adj\_list[i].push\_back(i);

//

// for (int j = 0; j < graph[i].size(); j++)

// {

// if (graph[i][j] == 1)

// {

// adj\_list[i].push\_back(j);

// }

// }

// }

// return adj\_list;

//}

//

//int searchResult(vector<int> arr, int k)

//{

// vector<int>::iterator it;

// it = find(arr.begin(), arr.end(), k);

//

// if (it != arr.end())

// {

// return (it - arr.begin());

// }

// else

// {

// return -1;

// }

//}

//

//int main()

//{

// int condition;

//

// cout << "Enter 1. for Adjacency Matrix\nEnter 2. for Adjacency List\nInput: ";

// cin >> condition;

//

// vector<vector<int>> adj;

// vector<int> causingVertices;

//

// if (condition == 1)

// {

// vector<vector<int>> matrix;

//

// int n;

// cout << "\n\t\t\*\*\*ADJACENCY MATRIX\*\*\*\n\n";

// cout << "Enter the number of vertices: ";

// cin >> n;

//

// cout << "\nEnter the Adjacency: " <<endl;

// for (int i = 0; i < n; i++)

// {

// vector<int> row;

// for (int j = 0; j < n; j++)

// {

// int value;

// cout << "\*Enter 1 for CONNECTED\* \n\*Enter 0 for NOT CONNECTED\*\n\n";

// cout << "Is the VERTEX " << j << " connected to " << i <<" : ";

// cin >> value;

// row.push\_back(value);

// }

//

// matrix.push\_back(row);

// }

//

// adj = matrixToAdjList(matrix);

//

// //vector<vector<int>> matrix =

// //{

// // // Project Test Case 1 : False

// // //1,2,3,4,5,6,7,8,9,10,11

// // /\*1\*/{0,0,1,0,1,0,0,1,0,0,0},

// // /\*2\*/{0,0,1,0,0,0,0,0,1,0,1},

// // /\*3\*/{1,1,0,1,0,0,0,0,0,0,0},

// // /\*4\*/{0,0,1,0,0,1,0,0,0,1,0},

// // /\*5\*/{1,0,0,0,0,1,1,0,0,0,0},

// // /\*6\*/{0,0,0,1,1,0,0,0,1,0,0},

// // /\*7\*/{0,0,0,0,1,0,0,0,0,1,0},

// // /\*8\*/{1,0,0,0,0,0,0,0,1,1,0},

// // /\*9\*/{0,1,0,0,0,1,0,1,0,0,0},

// // /\*10\*/{0,0,0,1,0,0,1,1,0,0,0},

// // /\*11\*/{0,1,0,0,0,0,1,0,0,0,0},

//

// // // Project Test Case 2 : True

// // //1,2,3,4,5,6,7,8,9,10,11

// // ///\*1\*/{0,1,0,0,0,0,0,0,1,1,0},

// // ///\*2\*/{1,0,1,0,0,1,0,0,0,0,0},

// // ///\*3\*/{0,1,0,1,0,0,0,1,0,0,0},

// // ///\*4\*/{0,0,1,0,1,0,0,0,0,0,1},

// // ///\*5\*/{0,0,0,1,0,1,0,0,1,0,0},

// // ///\*6\*/{0,1,0,0,1,0,1,0,0,0,0},

// // ///\*7\*/{0,0,0,0,0,1,0,1,0,0,1},

// // ///\*8\*/{0,0,1,0,0,0,1,0,1,0,0},

// // ///\*9\*/{1,0,0,0,1,0,0,1,0,0,0},

// // ///\*10\*/{1,0,0,0,0,0,0,0,0,0,1},

// // ///\*11\*/{0,0,0,1,0,0,1,0,0,1,0},

// //

// // // Test Case 1 : True

// // /\*{0, 1, 0, 1},

// // {1, 0, 1, 0},

// // {0, 1, 0, 1},

// // {1, 0, 1, 0}\*/

//

// // // Test Case 2 : False

// // /\*{0, 1, 1, 0},

// // {1, 0, 1, 0},

// // {1, 1, 0, 1},

// // {0, 0, 1, 0}\*/

//

// // // Test Case 3 : True

// // /\*{0, 1, 0, 1},

// // {1, 0, 1, 0},

// // {0, 1, 0, 1},

// // {1, 0, 1, 0},

// // {0, 0, 0, 0}\*/

//

// // // Test Case 4 : False

// // /\*{0, 1, 1, 0, 0},

// // {1, 0, 1, 0, 0},

// // {1, 1, 0, 1, 0},

// // {0, 0, 1, 0, 1},

// // {0, 0, 0, 1, 0}\*/

//

// //};

// //adj = matrixToAdjList(matrix);

// }

// else

// {

// vector<vector<int>> adjList;

//

// int n;

// cout << "\t\t\*\*\*ADJACENCY LIST\*\*\*\n\n";

// cout << "Enter the number of vertices for the Adjacency List : ";

// cin >> n;

//

// cout << "\nEnter the Adjacency : " << endl;

// for (int i = 0; i < n; i++)

// {

// vector<int> list;

// int m;

// cout << "Enter the number of Vertices adjacent to Vertex " << i + 1 << " : ";

// cin >> m;

// cout << "Enter the Vertices adjacent to Vertex " << i + 1 << " : ";

// for (int j = 0; j < m; j++)

// {

// int val;

// cin >> val;

// list.push\_back(val);

// }

//

// adjList.push\_back(list);

// }

//

// adj = adjList;

//

// //vector<vector<int>> adjList

// //{

// // /\*{1,3},

// // {0,2},

// // {1,3},

// // {0,2},\*/

//

// // /\*{0, 1},

// // {1, 2},

// // {2, 0}\*/

//

// // /\*{0,1},

// // {1,2},

// // {2,3},

// // {3,0}\*/

//

// // /\*{0, 5, 6, 7, 8},

// // {1, 6, 7, 8 ,9},

// // {2, 5, 6, 7, 9},

// // {3, 5, 6, 8, 9},

// // {4, 5, 7, 8, 9}\*/

//

// // /\*{0, 1, 1, 0, 0},

// // {1, 0, 1, 0, 0},

// // {1, 1, 0, 0, 1},

// // {0, 0, 0, 0, 1},

// // {0, 0, 1, 1, 0}\*/

//

// // /\*{1,2},

// // {0,2},

// // {0,1,3},

// // {2}\*/

//

// // /\*{1,2},

// // {0,2},

// // {0, 1,3},

// // {2,4},

// // {3}\*/

//

// // {0,1,2},

// // {1,0,2},

// // {2,0,1,3},

// // {3,2,4},

// // {4,3,5},

// // {5,4}

// //};

//

// }

//

// vector<int> arr1;

// vector<int> arr2;

// bool bipart = true;

//

// for (int i = 0; i < adj.size(); i++)

// {

// int indexMN1 = searchResult(arr1, adj[i][0]);

// int indexMN2 = searchResult(arr2, adj[i][0]);

//

// if (indexMN1 == -1 && indexMN2 == -1)

// {

// // Not Found the item

// arr1.push\_back(adj[i][0]);

// }

// else if (indexMN1 != -1 && indexMN2 != -1)

// {

// // Found the item

// bipart = false;

//

// if (searchResult(causingVertices, adj[i][0]) == -1)

// {

// causingVertices.push\_back(adj[i][0]);

// }

// }

//

// for (int j = 1; j < adj[i].size(); j++)

// {

// int indexMM1 = searchResult(arr1, adj[i][j]);

// int indexMM2 = searchResult(arr2, adj[i][j]);

//

// indexMN1 = searchResult(arr1, adj[i][0]);

// indexMN2 = searchResult(arr2, adj[i][0]);

//

// if (indexMM1 != -1 && indexMM2 != -1)

// {

// // Found the item

// bipart = false;

//

// if (searchResult(causingVertices, adj[i][j]) == -1)

// {

// causingVertices.push\_back(adj[i][j]);

// }

// }

// else if (indexMN1 != -1 || indexMN2 != -1)

// {

// // Found the item

//

// if (indexMN1 != -1 && indexMN2 == -1 && indexMM2 == -1)

// {

// arr2.push\_back(adj[i][j]);

// }

// else if (indexMN2 != -1 && indexMN1 == -1 && indexMM1 == -1)

// {

// arr1.push\_back(adj[i][j]);

// }

//

// indexMM1 = searchResult(arr1, adj[i][j]);

// indexMM2 = searchResult(arr2, adj[i][j]);

//

// if (indexMM1 != -1 && indexMM2 != -1)

// {

// bipart = false;

//

// if (searchResult(causingVertices, adj[i][j]) == -1)

// {

// causingVertices.push\_back(adj[i][j]);

// }

// }

// }

// else if (indexMM1 == -1 && indexMM2 == -1)

// {

// // Not Found the item

// arr2.push\_back(adj[i][j]);

// }

// }

// }

//

// cout << endl << "Vertix 1 List: ";

//

// for (int i = 0; i < arr1.size(); i++)

// {

// cout << arr1[i] << " ";

// }

// cout << endl << "Vertix 2 List: ";

//

// for (int i = 0; i < arr2.size(); i++)

// {

// cout << arr2[i] << " ";

// }

// cout << endl;

//

// cout << "\nIs Graph Bipartite: " << (bipart == 1? "Yes" : "No") << endl;

//

// if (bipart == false)

// {

// display(causingVertices);

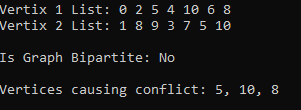
// }

//

// return 0;

//}

**Output**

****