МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

**Лабораторная работа №3.1**

по дисциплине: Дискретная математика

тема: «Отношения и их свойства»

Выполнил: ст. группы ПВ-221

Лоёк Никита Викторович

Проверили:

Бондаренко Татьяна Владимировна

Рязанов Юрий Дмитриевич

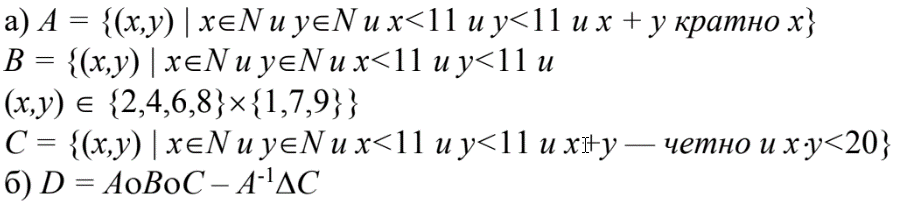
Белгород 2023 г.

Л а б о р а т о р н а я р а б о т а № 3.1

**Цель работы:** изучить способы задания отношений, операции над отношениями и

свойства отношений, научиться программно реализовывать операции и определять свойства отношений.

**Вариант 12**



**Задания**

**Часть 1. Операции над отношениями**

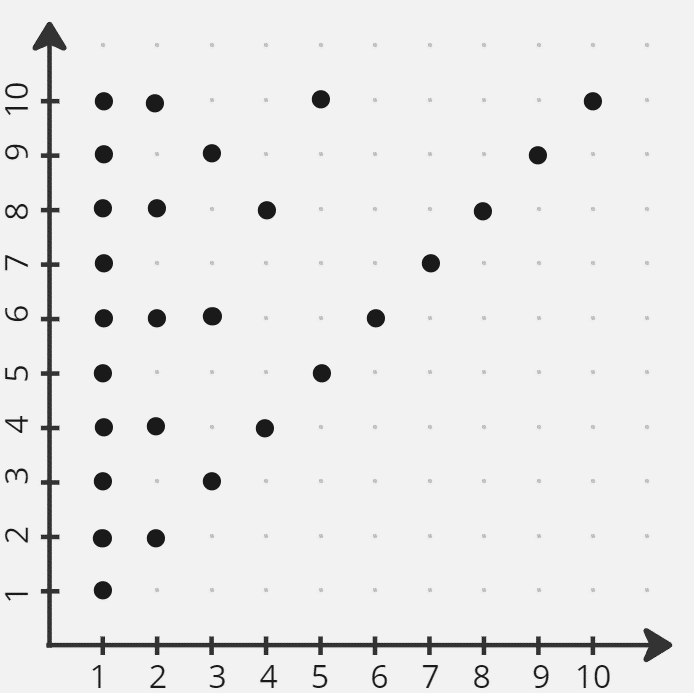
1.1. Представить отношения (см. Варианты заданий‖, п.а) графиком, графом и

матрицей.

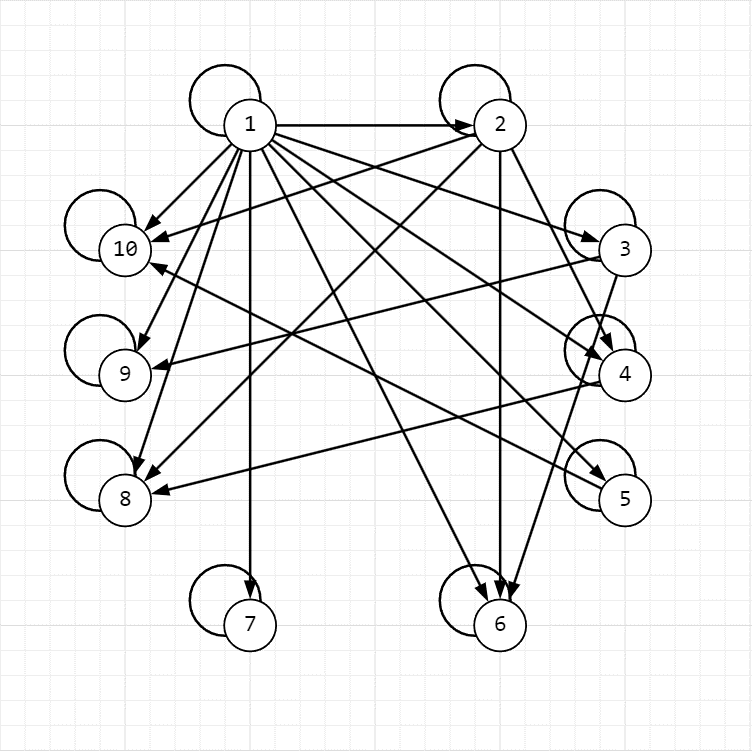
a)

Отношение :

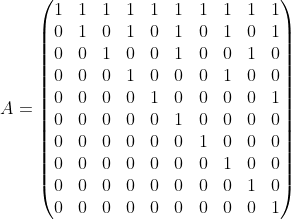
График:



Граф:

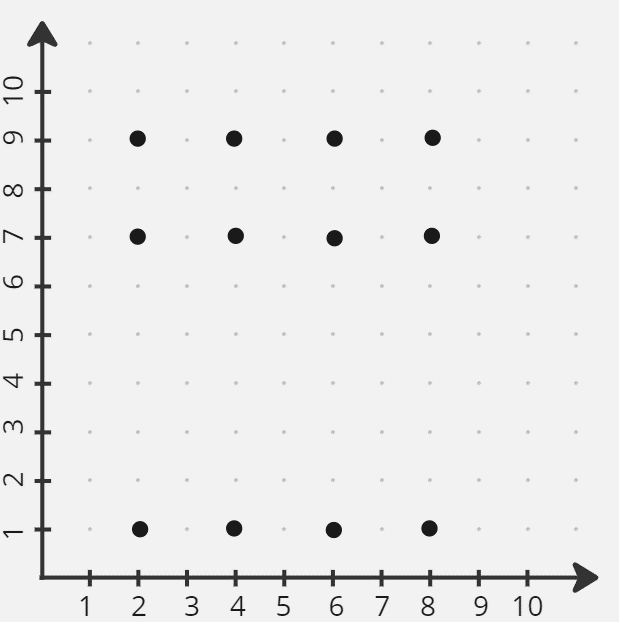


Матрица:

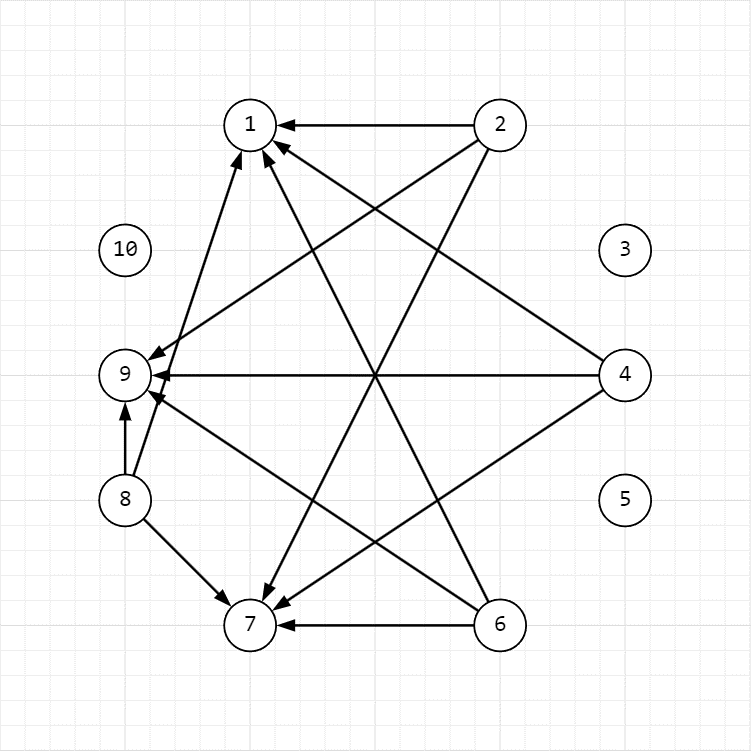


Отношение :

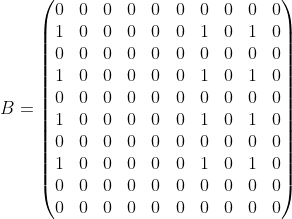
График:



Граф:

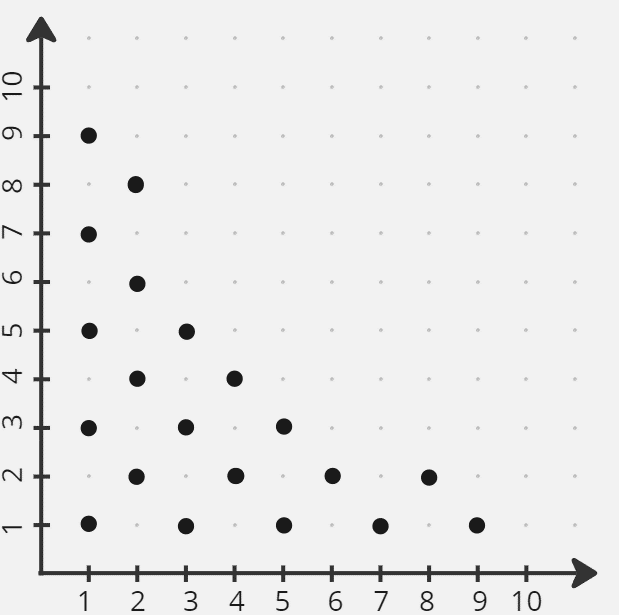


Матрица:

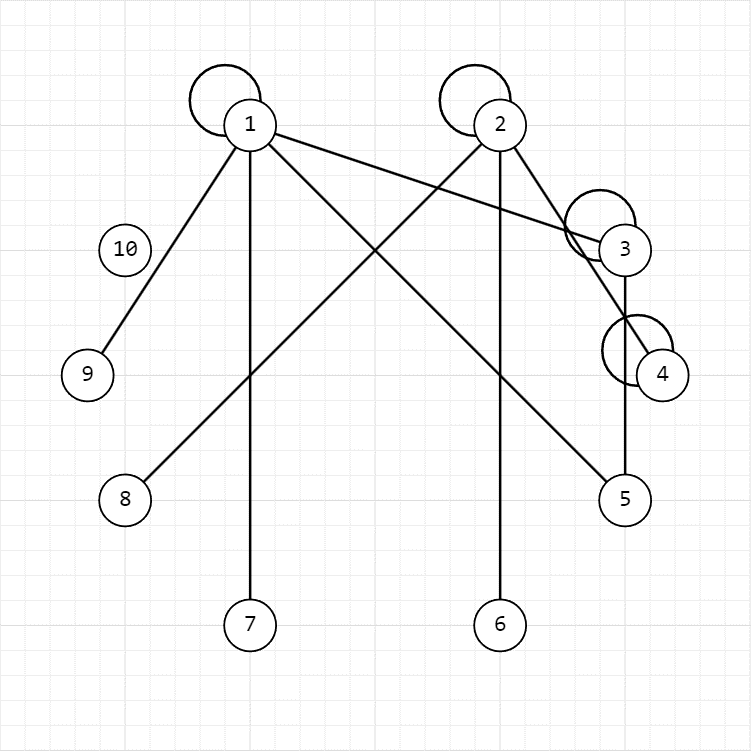


Отношение :

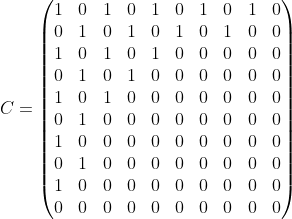
График:



Граф:



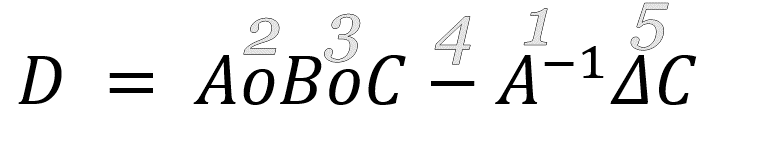
Матрица:



1.2. Вычислить значение выражения (см. ”Варианты заданий”, п.б) при заданных отношениях (см. ”Варианты заданий”, п.а).

б)

Решим по действиям:



A = {(1,1) (1,2) (1,3) (1,4) (1,5) (1,6) (1,7) (1,8) (1,9) (1,10) (2,2) (2,4) (2,6) (2,8) (2,10) (3,3) (3,6) (3,9) (4,4) (4,8) (5,5) (5,10) (6,6) (7,7) (8,8) (9,9) (10,10)}

B = {(2,1) (2,7) (2,9) (4,1) (4,7) (4,9) (6,1) (6,7) (6,9) (8,1) (8,7) (8,9)}

C = {(1,1) (1,3) (1,5) (1,7) (1,9) (2,2) (2,4) (2,6) (2,8) (3,1) (3,3) (3,5) (4,2) (4,4) (5,1) (5,3) (6,2) (7,1) (8,2) (9,1)}

1) = {(1,1) (2,1) (3,1) (4,1) (5,1) (6,1) (7,1) (8,1) (9,1) (10,1) (2,2) (4,2) (6,2) (8,2) (10,2) (3,3) (6,3) (9,3) (4,4) (8,4) (5,5) (10,5) (6,6) (7,7) (8,8) (9,9) (10,10)}

2) = {(1,1) (1,7) (1,9) (2,1) (2,7) (2,9) (3,1) (3,7) (3,9) (4,1) (4,7) (4,9) (6,1) (6,7) (6,9) (8,1) (8,7) (8,9)}

3) = {(1,1) (1,3) (1,5) (1,7) (1,9) (2,1) (2,3) (2,5) (2,7) (2,9) (3,1) (3,3) (3,5) (3,7) (3,9) (4,1) (4,3) (4,5) (4,7) (4,9) (6,1) (6,3) (6,5) (6,7) (6,9) (8,1) (8,3) (8,5) (8,7) (8,9)}

4) = {(1,3) (1,5) (1,7) (1,9) (2,3) (2,5) (2,7) (2,9) (3,5) (3,7) (3,9) (4,3) (4,5) (4,7) (4,9) (6,5) (6,7) (6,9) (8,3) (8,5) (8,7) (8,9)}

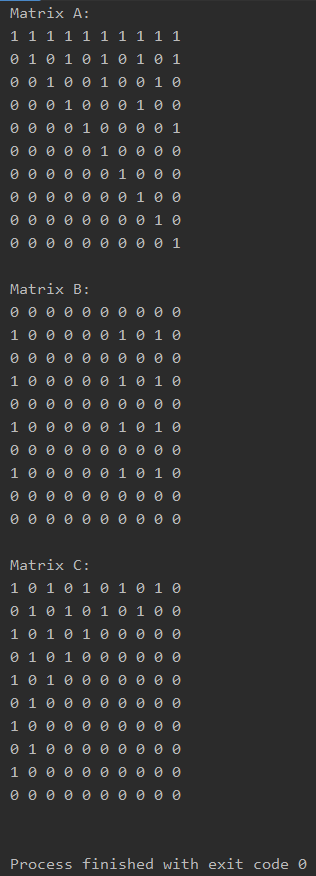
5) = {(1,1) (2,2) (2,3) (2,4) (2,5) (2,6) (2,7) (2,8) (2,9) (3,1) (3,3) (3,7) (3,9) (4,2) (4,3) (4,4) (4,5) (4,7) (4,9) (5,1) (5,2) (6,2) (6,5) (6,7) (6,9) (7,1) (8,2) (8,3) (8,5) (8,7) (8,9) (9,1)}

1.3. Написать программы, формирующие матрицы заданных отношений (см. ”Варианты заданий”, п.а).

a)

#include <iostream>  
#include <vector>  
  
  
using namespace std;  
  
  
void outputMatrix(vector<vector<bool>> &matrix) {  
 for (int i = 0; i < matrix.size(); i++) {  
 for (int j = 0; j < matrix[i].size(); j++) {  
 cout << matrix[i][j] << ' ';  
 }  
 cout << endl;  
 }  
 cout << endl;  
}  
  
  
void MatrixA(vector<vector<bool>> &matrix) {  
 for (int i = 0; i < matrix.size(); i++) {  
 for (int j = 0; j < matrix[i].size(); j++) {  
 int x = i + 1;  
 int y = j + 1;  
 if ((x + y) % x == 0) {  
 matrix[i][j] = true;  
 } else {  
 matrix[i][j] = false;  
 }  
 }  
 }  
 cout << "Matrix A:" << endl;  
 outputMatrix(matrix);  
}  
  
  
void MatrixB(vector<vector<bool>> &matrix) {  
 for (int i = 0; i < matrix.size(); i++) {  
 for (int j = 0; j < matrix[i].size(); j++) {  
 int x = i + 1;  
 int y = j + 1;  
 if ((x == 2 || x == 4 || x == 6 || x == 8) &&  
 (y == 1 || y == 7 || y == 9)) {  
 matrix[i][j] = true;  
 } else {  
 matrix[i][j] = false;  
 }  
 }  
 }  
 cout << "Matrix B:" << endl;  
 outputMatrix(matrix);  
}  
  
  
void MatrixC(vector<vector<bool>> &matrix) {  
 for (int i = 0; i < matrix.size(); i++) {  
 for (int j = 0; j < matrix[i].size(); j++) {  
 int x = i + 1;  
 int y = j + 1;  
 if ((x + y) % 2 == 0 && x \* y < 20) {  
 matrix[i][j] = true;  
 } else {  
 matrix[i][j] = false;  
 }  
 }  
 }  
 cout << "Matrix C:" << endl;  
 outputMatrix(matrix);  
}  
  
  
int main() {  
 vector<vector<bool>> matrix(10, vector<bool>(10));  
  
 MatrixA(matrix);  
 MatrixB(matrix);  
 MatrixC(matrix);  
  
 return 0;  
}

Результат работы программы:



1.4. Программно реализовать операции над отношениями.

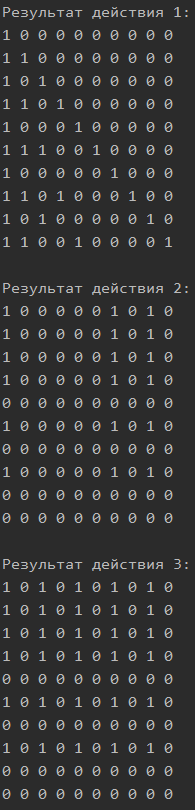
#include <iostream>  
#include <vector>  
#include <cassert>  
  
  
using namespace std;

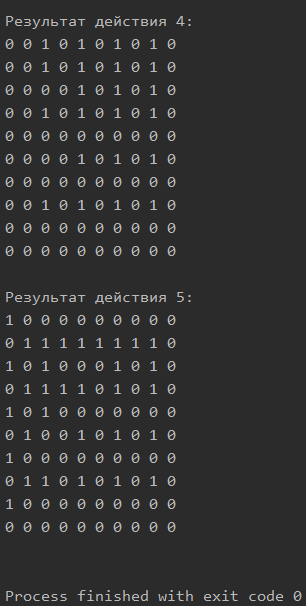
// равенство  
bool equality(vector<vector<bool>> &matrix1,  
 vector<vector<bool>> &matrix2) {  
 size\_t xSize = 0;  
 size\_t ySize = 0;  
 if ((matrix1.size() == matrix2.size()) && (matrix1[0].size() == matrix2[0].size())) {  
 xSize = matrix1.size();  
 ySize = matrix1[0].size();  
 } else {  
 assert(matrix1.size() == matrix2.size());  
 assert(matrix1[0].size() == matrix2[0].size());  
 }  
  
 for (int i = 0; i < xSize; i++) {  
 for (int j = 0; j < ySize; j++) {  
 if (matrix1[i][j] != matrix2[i][j]) {  
 return false;  
 }  
 }  
 }  
  
 return true;  
}  
  
// включение  
bool inclusion(vector<vector<bool>> &matrix1,  
 vector<vector<bool>> &matrix2) {  
 size\_t xSize = 0;  
 size\_t ySize = 0;  
 if ((matrix1.size() == matrix2.size()) && (matrix1[0].size() == matrix2[0].size())) {  
 xSize = matrix1.size();  
 ySize = matrix1[0].size();  
 } else {  
 assert(matrix1.size() == matrix2.size());  
 assert(matrix1[0].size() == matrix2[0].size());  
 }  
  
 for (int i = 0; i < xSize; i++) {  
 for (int j = 0; j < ySize; j++) {  
 if (matrix1[i][j] > matrix2[i][j]) {  
 return false;  
 }  
 }  
 }  
  
 return true;  
}  
  
  
// строгое включение  
bool strictInclusion(vector<vector<bool>> &matrix1,  
 vector<vector<bool>> &matrix2) {  
 size\_t xSize = 0;  
 size\_t ySize = 0;  
 if ((matrix1.size() == matrix2.size()) && (matrix1[0].size() == matrix2[0].size())) {  
 xSize = matrix1.size();  
 ySize = matrix1[0].size();  
 } else {  
 assert(matrix1.size() == matrix2.size());  
 assert(matrix1[0].size() == matrix2[0].size());  
 }  
  
 for (int i = 0; i < xSize; i++) {  
 for (int j = 0; j < ySize; j++) {  
 if (matrix1[i][j] >= matrix2[i][j]) {  
 return false;  
 }  
 }  
 }  
  
 return true;  
}  
  
  
// объединение  
void union\_(vector<vector<bool>> &matrix1,  
 vector<vector<bool>> &matrix2,  
 vector<vector<bool>> &resultMatrix) {  
 size\_t xSize = 0;  
 size\_t ySize = 0;  
 if ((matrix1.size() == matrix2.size() == resultMatrix.size()) &&  
 (matrix1[0].size() == matrix2[0].size() == resultMatrix[0].size())) {  
 xSize = matrix1.size();  
 ySize = matrix1[0].size();  
 } else {  
 assert(matrix1.size() == matrix2.size() == resultMatrix.size());  
 assert(matrix1[0].size() == matrix2[0].size() == resultMatrix[0].size());  
 }  
  
 for (int i = 0; i < xSize; i++) {  
 for (int j = 0; j < ySize; j++) {  
 resultMatrix[i][j] = matrix1[i][j] || matrix2[i][j];  
 }  
 }  
}  
  
  
// пересечение  
void intersection(vector<vector<bool>> &matrix1,  
 vector<vector<bool>> &matrix2,  
 vector<vector<bool>> &resultMatrix) {  
 size\_t xSize = 0;  
 size\_t ySize = 0;  
 if ((matrix1.size() == matrix2.size() == resultMatrix.size()) &&  
 (matrix1[0].size() == matrix2[0].size() == resultMatrix[0].size())) {  
 xSize = matrix1.size();  
 ySize = matrix1[0].size();  
 } else {  
 assert(matrix1.size() == matrix2.size() == resultMatrix.size());  
 assert(matrix1[0].size() == matrix2[0].size() == resultMatrix[0].size());  
 }  
  
 for (int i = 0; i < xSize; i++) {  
 for (int j = 0; j < ySize; j++) {  
 resultMatrix[i][j] = matrix1[i][j] && matrix2[i][j];  
 }  
 }  
}  
  
  
// разность  
void difference(vector<vector<bool>> &matrix1,  
 vector<vector<bool>> &matrix2,  
 vector<vector<bool>> &resultMatrix) {  
 size\_t xSize = 0;  
 size\_t ySize = 0;  
 if ((matrix1.size() == matrix2.size() == resultMatrix.size()) &&  
 (matrix1[0].size() == matrix2[0].size() == resultMatrix[0].size())) {  
 xSize = matrix1.size();  
 ySize = matrix1[0].size();  
 } else {  
 assert(matrix1.size() == matrix2.size() == resultMatrix.size());  
 assert(matrix1[0].size() == matrix2[0].size() == resultMatrix[0].size());  
 }  
  
 for (int i = 0; i < xSize; i++) {  
 for (int j = 0; j < ySize; j++) {  
 resultMatrix[i][j] = matrix1[i][j] && !matrix2[i][j];  
 }  
 }  
}  
  
  
// симметрическая разность  
void symmetricDifference(vector<vector<bool>> &matrix1,  
 vector<vector<bool>> &matrix2,  
 vector<vector<bool>> &resultMatrix) {  
 size\_t xSize = 0;  
 size\_t ySize = 0;  
 if ((matrix1.size() == matrix2.size() == resultMatrix.size()) &&  
 (matrix1[0].size() == matrix2[0].size() == resultMatrix[0].size())) {  
 xSize = matrix1.size();  
 ySize = matrix1[0].size();  
 } else {  
 assert(matrix1.size() == matrix2.size() == resultMatrix.size());  
 assert(matrix1[0].size() == matrix2[0].size() == resultMatrix[0].size());  
 }  
  
 for (int i = 0; i < xSize; i++) {  
 for (int j = 0; j < ySize; j++) {  
 resultMatrix[i][j] = matrix1[i][j] != matrix2[i][j];  
 }  
 }  
}  
  
  
// дополнение  
void addition(vector<vector<bool>> &matrix,  
 vector<vector<bool>> &resultMatrix) {  
  
 for (int i = 0; i < matrix.size(); i++) {  
 for (int j = 0; j < matrix[0].size(); j++) {  
 resultMatrix[i][j] = !matrix[i][j];  
 }  
 }  
}  
  
  
// обращение  
void appeal(vector<vector<bool>> &matrix,  
 vector<vector<bool>> &resultMatrix) {  
  
 for (int i = 0; i < matrix.size(); i++) {  
 for (int j = 0; j < matrix[0].size(); j++) {  
 resultMatrix[i][j] = matrix[j][i];  
 }  
 }  
}  
  
  
// композиция  
void composition(vector<vector<bool>> &matrix1,  
 vector<vector<bool>> &matrix2,  
 vector<vector<bool>> &resultMatrix) {  
 size\_t xSize = 0;  
 size\_t ySize = 0;  
 if ((matrix1.size() == matrix2.size()) &&  
 (matrix1[0].size() == matrix2[0].size())) {  
 xSize = matrix1.size();  
 ySize = matrix1[0].size();  
 } else {  
 assert(matrix1.size() == matrix2.size());  
 assert(matrix1[0].size() == matrix2[0].size());  
 }  
  
 for (int i = 0; i < xSize; i++) {  
 for (int j = 0; j < ySize; j++) {  
 bool flag = false;  
 for (int z = 0; !flag && z < max(xSize, ySize); z++) {  
 if (matrix1[i][z] && matrix2[z][j]) {  
 flag = true;  
 }  
 }  
 resultMatrix[i][j] = flag;  
 }  
 }  
}

1.5. Написать программу, вычисляющую значение выражения (см.“Варианты заданий”, п.б) и вычислить его при заданных отношениях (см.”Варианты заданий”, п.а).

#include <iostream>  
#include <vector>  
#include <cassert>  
#include <windows.h>  
  
  
using namespace std;  
  
  
void outputMatrix(vector<vector<bool>> &matrix) {  
 for (int i = 0; i < matrix.size(); i++) {  
 for (int j = 0; j < matrix[i].size(); j++) {  
 cout << matrix[i][j] << ' ';  
 }  
 cout << endl;  
 }  
 cout << endl;  
}  
  
  
void MatrixA(vector<vector<bool>> &matrix) {  
 for (int i = 0; i < matrix.size(); i++) {  
 for (int j = 0; j < matrix[i].size(); j++) {  
 int x = i + 1;  
 int y = j + 1;  
 if ((x + y) % x == 0) {  
 matrix[i][j] = true;  
 } else {  
 matrix[i][j] = false;  
 }  
 }  
 }  
}  
  
  
void MatrixB(vector<vector<bool>> &matrix) {  
 for (int i = 0; i < matrix.size(); i++) {  
 for (int j = 0; j < matrix[i].size(); j++) {  
 int x = i + 1;  
 int y = j + 1;  
 if ((x == 2 || x == 4 || x == 6 || x == 8) && (y == 1 || y == 7 || y == 9)) {  
 matrix[i][j] = true;  
 } else {  
 matrix[i][j] = false;  
 }  
 }  
 }  
}  
  
  
void MatrixC(vector<vector<bool>> &matrix) {  
 for (int i = 0; i < matrix.size(); i++) {  
 for (int j = 0; j < matrix[i].size(); j++) {  
 int x = i + 1;  
 int y = j + 1;  
 if ((x + y) % 2 == 0 && x \* y < 20) {  
 matrix[i][j] = true;  
 } else {  
 matrix[i][j] = false;  
 }  
 }  
 }  
}  
  
  
// равенство  
bool equality(vector<vector<bool>> &matrix1,  
 vector<vector<bool>> &matrix2) {  
 size\_t xSize = 0;  
 size\_t ySize = 0;  
 if ((matrix1.size() == matrix2.size()) && (matrix1[0].size() == matrix2[0].size())) {  
 xSize = matrix1.size();  
 ySize = matrix1[0].size();  
 } else {  
 assert(matrix1.size() == matrix2.size());  
 assert(matrix1[0].size() == matrix2[0].size());  
 }  
  
 for (int i = 0; i < xSize; i++) {  
 for (int j = 0; j < ySize; j++) {  
 if (matrix1[i][j] != matrix2[i][j]) {  
 return false;  
 }  
 }  
 }  
  
 return true;  
}  
  
// включение  
bool inclusion(vector<vector<bool>> &matrix1,  
 vector<vector<bool>> &matrix2) {  
 size\_t xSize = 0;  
 size\_t ySize = 0;  
 if ((matrix1.size() == matrix2.size()) && (matrix1[0].size() == matrix2[0].size())) {  
 xSize = matrix1.size();  
 ySize = matrix1[0].size();  
 } else {  
 assert(matrix1.size() == matrix2.size());  
 assert(matrix1[0].size() == matrix2[0].size());  
 }  
  
 for (int i = 0; i < xSize; i++) {  
 for (int j = 0; j < ySize; j++) {  
 if (matrix1[i][j] > matrix2[i][j]) {  
 return false;  
 }  
 }  
 }  
  
 return true;  
}  
  
  
// строгое включение  
bool strictInclusion(vector<vector<bool>> &matrix1,  
 vector<vector<bool>> &matrix2) {  
 size\_t xSize = 0;  
 size\_t ySize = 0;  
 if ((matrix1.size() == matrix2.size()) && (matrix1[0].size() == matrix2[0].size())) {  
 xSize = matrix1.size();  
 ySize = matrix1[0].size();  
 } else {  
 assert(matrix1.size() == matrix2.size());  
 assert(matrix1[0].size() == matrix2[0].size());  
 }  
  
 for (int i = 0; i < xSize; i++) {  
 for (int j = 0; j < ySize; j++) {  
 if (matrix1[i][j] >= matrix2[i][j]) {  
 return false;  
 }  
 }  
 }  
  
 return true;  
}  
  
  
// объединение  
void union\_(vector<vector<bool>> &matrix1,  
 vector<vector<bool>> &matrix2,  
 vector<vector<bool>> &resultMatrix) {  
 size\_t xSize = 0;  
 size\_t ySize = 0;  
 if ((matrix1.size() == matrix2.size() == resultMatrix.size()) &&  
 (matrix1[0].size() == matrix2[0].size() == resultMatrix[0].size())) {  
 xSize = matrix1.size();  
 ySize = matrix1[0].size();  
 } else {  
 assert(matrix1.size() == matrix2.size() == resultMatrix.size());  
 assert(matrix1[0].size() == matrix2[0].size() == resultMatrix[0].size());  
 }  
  
 for (int i = 0; i < xSize; i++) {  
 for (int j = 0; j < ySize; j++) {  
 resultMatrix[i][j] = matrix1[i][j] || matrix2[i][j];  
 }  
 }  
}  
  
  
// пересечение  
void intersection(vector<vector<bool>> &matrix1,  
 vector<vector<bool>> &matrix2,  
 vector<vector<bool>> &resultMatrix) {  
 size\_t xSize = 0;  
 size\_t ySize = 0;  
 if ((matrix1.size() == matrix2.size() == resultMatrix.size()) &&  
 (matrix1[0].size() == matrix2[0].size() == resultMatrix[0].size())) {  
 xSize = matrix1.size();  
 ySize = matrix1[0].size();  
 } else {  
 assert(matrix1.size() == matrix2.size() == resultMatrix.size());  
 assert(matrix1[0].size() == matrix2[0].size() == resultMatrix[0].size());  
 }  
  
 for (int i = 0; i < xSize; i++) {  
 for (int j = 0; j < ySize; j++) {  
 resultMatrix[i][j] = matrix1[i][j] && matrix2[i][j];  
 }  
 }  
}  
  
  
// разность  
void difference(vector<vector<bool>> &matrix1,  
 vector<vector<bool>> &matrix2,  
 vector<vector<bool>> &resultMatrix) {  
 size\_t xSize = 0;  
 size\_t ySize = 0;  
 if ((matrix1.size() == matrix2.size()) &&  
 (matrix1[0].size() == matrix2[0].size())) {  
 xSize = matrix1.size();  
 ySize = matrix1[0].size();  
 } else {  
 assert(matrix1.size() == matrix2.size());  
 assert(matrix1[0].size() == matrix2[0].size());  
 }  
  
 for (int i = 0; i < xSize; i++) {  
 for (int j = 0; j < ySize; j++) {  
 resultMatrix[i][j] = matrix1[i][j] && !matrix2[i][j];  
 }  
 }  
}  
  
  
// симметрическая разность  
void symmetricDifference(vector<vector<bool>> &matrix1,  
 vector<vector<bool>> &matrix2,  
 vector<vector<bool>> &resultMatrix) {  
 size\_t xSize = 0;  
 size\_t ySize = 0;  
 if ((matrix1.size() == matrix2.size()) &&  
 (matrix1[0].size() == matrix2[0].size())) {  
 xSize = matrix1.size();  
 ySize = matrix1[0].size();  
 } else {  
 assert(matrix1.size() == matrix2.size());  
 assert(matrix1[0].size() == matrix2[0].size());  
 }  
  
 for (int i = 0; i < xSize; i++) {  
 for (int j = 0; j < ySize; j++) {  
 resultMatrix[i][j] = matrix1[i][j] != matrix2[i][j];  
 }  
 }  
}  
  
  
// дополнение  
void addition(vector<vector<bool>> &matrix,  
 vector<vector<bool>> &resultMatrix) {  
  
 for (int i = 0; i < matrix.size(); i++) {  
 for (int j = 0; j < matrix[0].size(); j++) {  
 resultMatrix[i][j] = !matrix[i][j];  
 }  
 }  
}  
  
  
// обращение  
void appeal(vector<vector<bool>> &matrix,  
 vector<vector<bool>> &resultMatrix) {  
  
 for (int i = 0; i < matrix.size(); i++) {  
 for (int j = 0; j < matrix[0].size(); j++) {  
 resultMatrix[i][j] = matrix[j][i];  
 }  
 }  
}  
  
  
// композиция  
void composition(vector<vector<bool>> &matrix1,  
 vector<vector<bool>> &matrix2,  
 vector<vector<bool>> &resultMatrix) {  
 size\_t xSize = 0;  
 size\_t ySize = 0;  
 if ((matrix1.size() == matrix2.size()) &&  
 (matrix1[0].size() == matrix2[0].size())) {  
 xSize = matrix1.size();  
 ySize = matrix1[0].size();  
 } else {  
 assert(matrix1.size() == matrix2.size());  
 assert(matrix1[0].size() == matrix2[0].size());  
 }  
  
 for (int i = 0; i < xSize; i++) {  
 for (int j = 0; j < ySize; j++) {  
 bool flag = false;  
 for (int z = 0; !flag && z < max(xSize, ySize); z++) {  
 if (matrix1[i][z] && matrix2[z][j]) {  
 flag = true;  
 }  
 }  
 resultMatrix[i][j] = flag;  
 }  
 }  
}  
  
  
int main() {  
 SetConsoleOutputCP(CP\_UTF8);  
  
 vector<vector<bool>> matrixA(10, vector<bool>(10));  
 MatrixA(matrixA);  
 vector<vector<bool>> matrixB(10, vector<bool>(10));  
 MatrixB(matrixB);  
 vector<vector<bool>> matrixC(10, vector<bool>(10));  
 MatrixC(matrixC);  
  
 vector<vector<bool>> \_1(10, vector<bool>(10));  
 appeal(matrixA, \_1);  
  
 cout << "Результат действия 1:" << endl;  
 outputMatrix(\_1);  
  
 vector<vector<bool>> \_2(10, vector<bool>(10));  
 composition(matrixA, matrixB, \_2);  
  
 cout << "Результат действия 2:" << endl;  
 outputMatrix(\_2);  
  
 vector<vector<bool>> \_3(10, vector<bool>(10));  
 composition(\_2, matrixC, \_3);  
  
 cout << "Результат действия 3:" << endl;  
 outputMatrix(\_3);  
  
 vector<vector<bool>> \_4(10, vector<bool>(10));  
 difference(\_3, \_1, \_4);  
  
 cout << "Результат действия 4:" << endl;  
 outputMatrix(\_4);  
  
 vector<vector<bool>> \_5(10, vector<bool>(10));  
 symmetricDifference(\_4, matrixC, \_5);  
  
 cout << "Результат действия 5:" << endl;  
 outputMatrix(\_5);  
   
 return 0;  
}

Результат работы программы:





**Часть 2. Свойства отношений**

2.1. Определить основные свойства отношений (см. ”Варианты заданий”, п.а).

a)

Отношение A:

Рефлексивно

Не антирефлексивно, т. к. по индексу (0, 0) находится 1

Не симметрично, т. к. по индексам (0, 1) и (1, 0) находятся 1 и 0

Антисимметрично

Транзитивно

Не антитранзитивно, т. к. по индексам (0, 0), (0, 0) и (0, 0) находятся 1, 1 и 1

Не полно, т. к. по индексам (1, 2) и (2, 1) находятся 0 и 0

Отношение B:

Не рефлексивно, т. к. по индексу (0, 0) находится 0

Антирефлексивно

Не симметрично, т. к. по индексам (0, 1) и (1, 0) находятся 0 и 1

Антисимметрично

Транзитивно

Антитранзитивно

Не полно, т. к. по индексам (0, 2) и (2, 0) находятся 0 и 0

Отношение C:

Не рефлексивно, т. к. по индексу (4, 4) находится 0

Не антирефлексивно, т. к. по индексу (0, 0) находится 1

Симметрично

Не антисимметрично, т. к. по индексам (0, 2) и (2, 0) находятся 1 и 1

Не транзитивно, т. к. по индексам (2, 0), (0, 6) и (2, 6) находятся 1, 1 и 0

Не антитранзитивно, т. к. по индексам (0, 0), (0, 0) и (0, 0) находятся 1, 1 и 1

Не полно, т. к. по индексам (0, 1) и (1, 0) находятся 0 и 0

2.2. Определить, являются ли заданные отношения отношениями толерантности, эквивалентности и порядка.

Отношение A:

Не является отношением толерантности, эквивалентности

Является отношением порядка, нестрогого порядка

Отношение B:

Не является отношением толерантности, эквивалентности

Является отношением порядка, строгого порядка

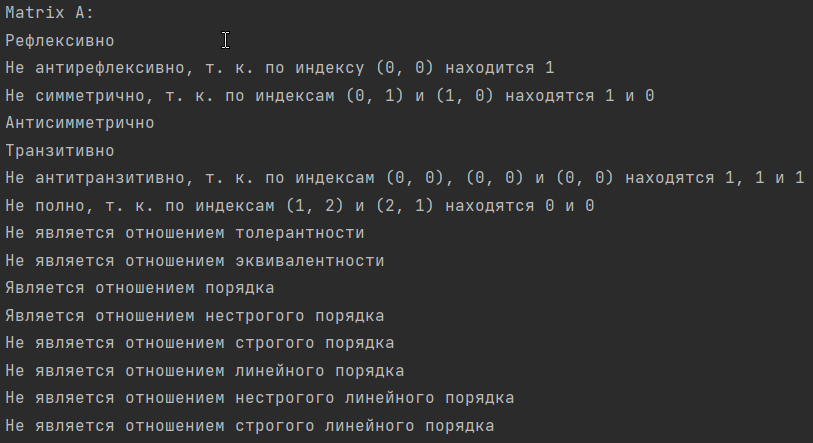
Отношение C:

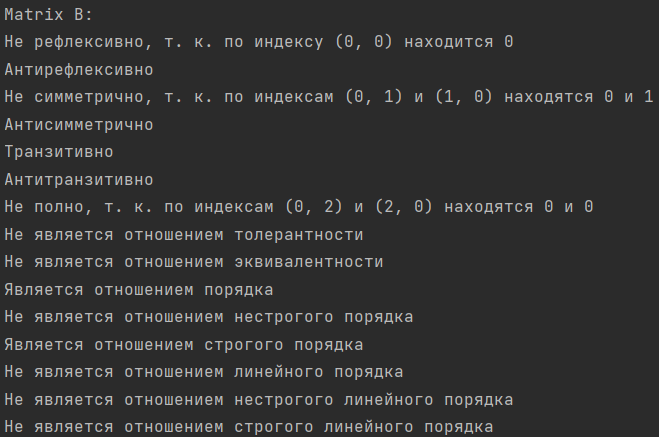
Не является отношением толерантности, эквивалентности, порядка

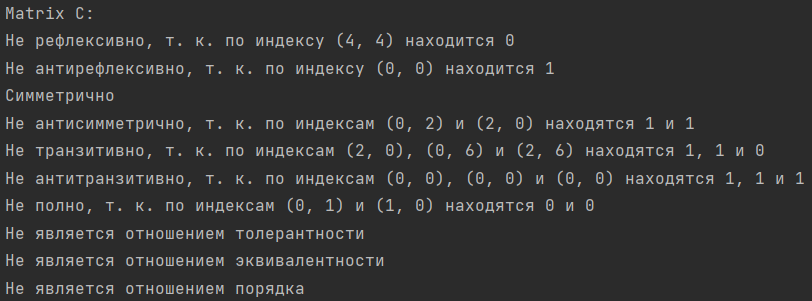
2.3. Написать программу, определяющую свойства отношения, в том числе толерантности, эквивалентности и порядка, и определить свойства отношений (см. ”Варианты заданий”, п.а).

#include <iostream>  
#include <vector>  
#include <windows.h>  
  
  
using namespace std;  
  
  
void MatrixA(vector <vector<bool>> &matrix) {  
 for (int i = 0; i < matrix.size(); i++) {  
 for (int j = 0; j < matrix[i].size(); j++) {  
 int x = i + 1;  
 int y = j + 1;  
 if ((x + y) % x == 0) {  
 matrix[i][j] = true;  
 } else {  
 matrix[i][j] = false;  
 }  
 }  
 }  
}  
  
  
void MatrixB(vector <vector<bool>> &matrix) {  
 for (int i = 0; i < matrix.size(); i++) {  
 for (int j = 0; j < matrix[i].size(); j++) {  
 int x = i + 1;  
 int y = j + 1;  
 if ((x == 2 || x == 4 || x == 6 || x == 8) &&  
 (y == 1 || y == 7 || y == 9)) {  
 matrix[i][j] = true;  
 } else {  
 matrix[i][j] = false;  
 }  
 }  
 }  
}  
  
  
void MatrixC(vector <vector<bool>> &matrix) {  
 for (int i = 0; i < matrix.size(); i++) {  
 for (int j = 0; j < matrix[i].size(); j++) {  
 int x = i + 1;  
 int y = j + 1;  
 if ((x + y) % 2 == 0 && x \* y < 20) {  
 matrix[i][j] = true;  
 } else {  
 matrix[i][j] = false;  
 }  
 }  
 }  
}  
  
  
vector <vector<int>> reflectivity\_(vector <vector<bool>> &matrix) {  
 for (int i = 0; i < matrix.size(); i++) {  
 if (!matrix[i][i]) {  
 return vector < vector < int >> **{**{i, i}**}**;  
 }  
 }  
 return vector < vector < int >> {};  
}  
  
  
void reflectivity(vector <vector<bool>> &matrix) {  
 vector <vector<int>> reflectivity = reflectivity\_(matrix);  
 if (reflectivity.empty()) {  
 cout << "Рефлексивно" << endl;  
 } else {  
 int i = reflectivity[0][0];  
 int j = reflectivity[0][1];  
  
 cout << "Не рефлексивно, т. к. по индексу (" << i << ", " << j << ") находится " << matrix[i][j] << endl;  
 }  
}  
  
  
vector <vector<int>> antireflectivity\_(vector <vector<bool>> &matrix) {  
 for (int i = 0; i < matrix.size(); i++) {  
 if (matrix[i][i]) {  
 return vector < vector < int >> **{**{i, i}**}**;  
 }  
 }  
 return vector < vector < int >> {};  
}  
  
  
void antireflectivity(vector <vector<bool>> &matrix) {  
 vector <vector<int>> antireflectivity = antireflectivity\_(matrix);  
  
 if (antireflectivity.empty()) {  
 cout << "Антирефлексивно" << endl;  
 } else {  
 int i = antireflectivity[0][0];  
 int j = antireflectivity[0][1];  
  
 cout << "Не антирефлексивно, т. к. по индексу (" << i << ", " << j << ") находится " << matrix[i][j] << endl;  
 }  
}  
  
  
vector <vector<int>> symmetry\_(vector <vector<bool>> &matrix) {  
 for (int i = 0; i < matrix.size(); i++) {  
 for (int j = i + 1; j < matrix[0].size(); j++) {  
 if (matrix[i][j] != matrix[j][i]) {  
 return vector < vector < int >> **{**{i, j},  
 {j, i}**}**;  
 }  
 }  
 }  
 return vector < vector < int >> {};  
}  
  
  
void symmetry(vector <vector<bool>> &matrix) {  
 vector <vector<int>> symmetry = symmetry\_(matrix);  
 if (symmetry.empty()) {  
 cout << "Симметрично" << endl;  
 } else {  
 int i1 = symmetry[0][0];  
 int j1 = symmetry[0][1];  
 int i2 = symmetry[1][0];  
 int j2 = symmetry[1][1];  
 cout << "Не симметрично, т. к. по индексам (" << i1 << ", " << j1 << ") и ("  
 << i2 << ", " << j2 << ") находятся " << matrix[i1][j1] << " и " << matrix[i2][j2] << endl;  
 }  
}  
  
  
vector <vector<int>> antisymmetry\_(vector <vector<bool>> &matrix) {  
 for (int i = 0; i < matrix.size(); i++) {  
 for (int j = i + 1; j < matrix[0].size(); j++) {  
 if (matrix[i][j] && matrix[j][i]) {  
 return vector < vector < int >> **{**{i, j},  
 {j, i}**}**;  
 }  
 }  
 }  
 return vector < vector < int >> {};  
}  
  
  
void antisymmetry(vector <vector<bool>> &matrix) {  
 vector <vector<int>> antisymmetry = antisymmetry\_(matrix);  
 if (antisymmetry.empty()) {  
 cout << "Антисимметрично" << endl;  
 } else {  
 int i1 = antisymmetry[0][0];  
 int j1 = antisymmetry[0][1];  
 int i2 = antisymmetry[1][0];  
 int j2 = antisymmetry[1][1];  
 cout << "Не антисимметрично, т. к. по индексам (" << i1 << ", " << j1 << ") и ("  
 << i2 << ", " << j2 << ") находятся " << matrix[i1][j1] << " и " << matrix[i2][j2] << endl;  
 }  
}  
  
  
vector <vector<int>> transitivity\_(vector <vector<bool>> &matrix) {  
 for (int i = 0; i < matrix.size(); i++) {  
 for (int j = 0; j < matrix[0].size(); j++) {  
 for (int z = 0; z < matrix.size(); z++) {  
 if (matrix[i][z] && matrix[z][j]) {  
 if (!matrix[i][j]) {  
 return vector < vector < int >> **{**{i, z},  
 {z, j},  
 {i, j}**}**;  
 }  
 }  
 }  
 }  
 }  
  
 return vector < vector < int >> {};  
}  
  
  
void transitivity(vector <vector<bool>> &matrix) {  
 vector <vector<int>> transitivity = transitivity\_(matrix);  
 if (transitivity.empty()) {  
 cout << "Транзитивно" << endl;  
 } else {  
 int i1 = transitivity[0][0];  
 int j1 = transitivity[0][1];  
 int i2 = transitivity[1][0];  
 int j2 = transitivity[1][1];  
 int i3 = transitivity[2][0];  
 int j3 = transitivity[2][1];  
 cout << "Не транзитивно, т. к. по индексам (" << i1 << ", " << j1 << "), ("  
 << i2 << ", " << j2 << ") и (" << i3 << ", " << j3 << ") находятся " << matrix[i1][j1] << ", "  
 << matrix[i2][j2] << " и " << matrix[i3][j3] << endl;  
  
 }  
}  
  
  
vector <vector<int>> antitransitivity\_(vector <vector<bool>> &matrix) {  
 for (int i = 0; i < matrix.size(); i++) {  
 for (int j = 0; j < matrix[0].size(); j++) {  
 for (int z = 0; z < matrix.size(); z++) {  
 if (matrix[i][z] && matrix[z][j]) {  
 if (matrix[i][j]) {  
 return vector < vector < int >> **{**{i, z},  
 {z, j},  
 {i, j}**}**;  
 }  
 }  
 }  
 }  
 }  
 return vector < vector < int >> {};  
}  
  
  
void antitransitivity(vector <vector<bool>> &matrix) {  
 vector <vector<int>> antitransitivity = antitransitivity\_(matrix);  
 if (antitransitivity.empty()) {  
 cout << "Антитранзитивно" << endl;  
 } else {  
 int i1 = antitransitivity[0][0];  
 int j1 = antitransitivity[0][1];  
 int i2 = antitransitivity[1][0];  
 int j2 = antitransitivity[1][1];  
 int i3 = antitransitivity[2][0];  
 int j3 = antitransitivity[2][1];  
 cout << "Не антитранзитивно, т. к. по индексам (" << i1 << ", " << j1 << "), ("  
 << i2 << ", " << j2 << ") и (" << i3 << ", " << j3 << ") находятся " << matrix[i1][j1] << ", "  
 << matrix[i2][j2] << " и " << matrix[i3][j3] << endl;  
 }  
}  
  
  
vector <vector<int>> completeness\_(vector <vector<bool>> &matrix) {  
 for (int i = 0; i < matrix.size(); i++) {  
 for (int j = i + 1; j < matrix[0].size(); j++) {  
 if (!matrix[i][j] && !matrix[j][i]) {  
 return vector < vector < int >> **{**{i, j},  
 {j, i}**}**;  
 }  
 }  
 }  
 return vector < vector < int >> {};  
}  
  
  
void completeness(vector <vector<bool>> &matrix) {  
 vector <vector<int>> completeness = completeness\_(matrix);  
 if (completeness.empty()) {  
 cout << "Полно" << endl;  
 } else {  
 int i1 = completeness[0][0];  
 int j1 = completeness[0][1];  
 int i2 = completeness[1][0];  
 int j2 = completeness[1][1];  
 cout << "Не полно, т. к. по индексам (" << i1 << ", " << j1 << ") и ("  
 << i2 << ", " << j2 << ") находятся " << matrix[i1][j1] << " и " << matrix[i2][j2] << endl;  
 }  
}  
  
  
bool tolerance\_(vector <vector<bool>> &matrix) {  
 return reflectivity\_(matrix).empty() && symmetry\_(matrix).empty();  
}  
  
  
void tolerance(vector <vector<bool>> &matrix) {  
 if (tolerance\_(matrix)) {  
 cout << "Является отношением толерантности" << endl;  
 } else {  
 cout << "Не является отношением толерантности" << endl;  
 }  
}  
  
  
bool equivalence\_(vector <vector<bool>> &matrix) {  
 return reflectivity\_(matrix).empty() && symmetry\_(matrix).empty() && transitivity\_(matrix).empty();  
}  
  
  
void equivalence(vector <vector<bool>> &matrix) {  
 if (equivalence\_(matrix)) {  
 cout << "Является отношением эквивалентности" << endl;  
 } else {  
 cout << "Не является отношением эквивалентности" << endl;  
 }  
}  
  
  
bool order\_(vector <vector<bool>> &matrix) {  
 return antisymmetry\_(matrix).empty() && transitivity\_(matrix).empty();  
}  
  
  
void order(vector <vector<bool>> &matrix) {  
 if (order\_(matrix)) {  
 cout << "Является отношением порядка" << endl;  
 } else {  
 cout << "Не является отношением порядка" << endl;  
 }  
}  
  
  
bool notStrictOrder\_(vector <vector<bool>> &matrix) {  
 return order\_(matrix) && reflectivity\_(matrix).empty();  
}  
  
  
void notStrictOrder(vector <vector<bool>> &matrix) {  
 if (notStrictOrder\_(matrix)) {  
 cout << "Является отношением нестрогого порядка" << endl;  
 } else {  
 cout << "Не является отношением нестрогого порядка" << endl;  
 }  
}  
  
  
bool strictOrder\_(vector <vector<bool>> &matrix) {  
 return order\_(matrix) && antireflectivity\_(matrix).empty();  
}  
  
  
void strictOrder(vector <vector<bool>> &matrix) {  
 if (strictOrder\_(matrix)) {  
 cout << "Является отношением строгого порядка" << endl;  
 } else {  
 cout << "Не является отношением строгого порядка" << endl;  
 }  
}  
  
  
bool linearOrder\_(vector <vector<bool>> &matrix) {  
 return order\_(matrix) && completeness\_(matrix).empty();  
}  
  
  
void linearOrder(vector <vector<bool>> &matrix) {  
 if (linearOrder\_(matrix)) {  
 cout << "Является отношением линейного порядка" << endl;  
 } else {  
 cout << "Не является отношением линейного порядка" << endl;  
 }  
}  
  
  
bool notStrictLinearOrder\_(vector <vector<bool>> &matrix) {  
 return notStrictOrder\_(matrix) && completeness\_(matrix).empty();  
}  
  
  
void notStrictLinearOrder(vector <vector<bool>> &matrix) {  
 if (notStrictLinearOrder\_(matrix)) {  
 cout << "Является отношением нестрогого линейного порядка" << endl;  
 } else {  
 cout << "Не является отношением нестрогого линейного порядка" << endl;  
 }  
}  
  
  
bool strictLinearOrder\_(vector <vector<bool>> &matrix) {  
 return strictOrder\_(matrix) && completeness\_(matrix).empty();  
}  
  
  
void strictLinearOrder(vector <vector<bool>> &matrix) {  
 if (strictLinearOrder\_(matrix)) {  
 cout << "Является отношением строгого линейного порядка" << endl;  
 } else {  
 cout << "Не является отношением строгого линейного порядка" << endl;  
 }  
}  
  
  
void defProperties(vector <vector<bool>> &matrix) {  
 reflectivity(matrix);  
 antireflectivity(matrix);  
 symmetry(matrix);  
 antisymmetry(matrix);  
 transitivity(matrix);  
 antitransitivity(matrix);  
 completeness(matrix);  
  
 tolerance(matrix);  
 equivalence(matrix);  
 order(matrix);  
 if (order\_(matrix)) {  
 notStrictOrder(matrix);  
 strictOrder(matrix);  
 linearOrder(matrix);  
 notStrictLinearOrder(matrix);  
 strictLinearOrder(matrix);  
 }  
 cout << endl;  
}  
  
  
int main() {  
 SetConsoleOutputCP(CP\_UTF8);  
  
 vector <vector<bool>> matrixA(10, vector<bool>(10));  
 MatrixA(matrixA);  
 vector <vector<bool>> matrixB(10, vector<bool>(10));  
 MatrixB(matrixB);  
 vector <vector<bool>> matrixC(10, vector<bool>(10));  
 MatrixC(matrixC);  
  
 cout << "Matrix A:" << endl;  
 defProperties(matrixA);  
 cout << "Matrix B:" << endl;  
 defProperties(matrixB);  
 cout << "Matrix C:" << endl;  
 defProperties(matrixC);  
  
 return 0;  
}

Результат работы программы:







**Вывод**

**Вывод**: в ходе работы были изучены способы задания отношений, операции над отношениями и свойства отношений, программно реализованы операции и определены свойства отношений.