#include<iostream>

#include<clocale>

using namespace std;

int\*\* PereChislenie(int key);

int\* Z(int left, int right); // заполнение побочных множеств

void Watch(int \*\*M, int mM);

void WatchMn(int \*M, int mM);

void KonZap();

void menu();

int\*\* InOne(int \*\*A, int \*\*B, int mA, int mB); // Объединение А и В

int\* InOneMn(int \*A, int \*B, int mA, int mB); // Объединение множеств

int\*\* Peresek(int \*\*A, int \*\*B, int mA, int mB); // Пересечение А и В

int\* PeresekMn(int \*A, int\*B, int mA, int mB); // Пересечение множеств

int\*\* Pa3HocTb(int \*\*U, int \*\*W, int mU, int mW); // Разность А и В

int\* Pa3HocTbMn(int \*U, int \*W, int mU, int mW); // Разность множеств

int\*\* SimRas(int \*\*A, int \*\*B, int mA, int mB); // Симметрическая разность A u B

int\* SimRasMn(int \*A, int \*B, int mA, int mB); // Симметрическая разность множеств

int\*\* Inversion(int \*\*M, int mM); // Инверсия

int\*\* Exposition(int \*\*A, int \*\*B, int mA, int mB); // Композиция соответствий

int\* Obraz(int \*\*G, int mG); // Образ

int\* ProObraz(int \*\*G, int mG); // ПроОбраз

int\*\* Suzko(int \*Y, int \*\*G, int mG); // Сужение

int\*\* Dekart(int \*A, int \*B, int mA, int mB, int mD);// Декартово произведение множеств

int\*\* Continue(int \*\*G, int mG); // Продолжение соответствия

int main()

{

setlocale(LC\_ALL, "rus");

int \*X = Z(1, 10); int \*Y = Z(5, 15); // для соответствия А

int \*U = Z(15, 25); int \*V = Z(10, 20); // для соответствия В

cout << "Область отправления соответствия А:\t"; WatchMn(X, X[0]); cout << endl;

cout << "Область прибытия соответствия А:\t"; WatchMn(Y, Y[0]); cout << endl;

cout << "Введите мощность графика G соответствия А:\t";

int mG; // 1 Пользователь вводит мощность n графика G первого соответствия A.

cin >> mG;

int \*\*G = PereChislenie(mG); // 2 Пользователь последовательно вводит n пар графика G соответствия A.

cout << "Область отправления соответствия B:\t"; WatchMn(U, U[0]); cout << endl;

cout << "Область прибытия соответствия B:\t"; WatchMn(V, V[0]); cout << endl;

cout << "Введите мощность графика F соответствия B:\t";

int mF;

cin >> mF; // 3 Пользователь вводит мощность m графика F второго соответствия B.

int \*\*F = PereChislenie(mF); // 4 Пользователь последовательно вводит m пар графика соответствия B.

KonZap(); //5 Заполняются множества, указанные в уточнении постановки задачи :

//5.1 Область отправления Х соответствия А заполняется натуральными числами от 1 до 10.

// 5.2 Область прибытия Y соответствия А заполняется натуральными числами от 5 до 15.

// 5.3 Область отправления U соответствия B заполняется натуральными числами от 15 до 25.

// 5.4 Область прибытия V соответствия B заполняется натуральными числами от 10 до 20.

while (true)

{

system("cls");

cout << "A = <"; WatchMn(X, X[0]); cout << ", "; WatchMn(Y, Y[0]); cout << ", "; Watch(G, mG); cout << ">" << endl;

cout << "B = <"; WatchMn(U, U[0]); cout << ", "; WatchMn(V, V[0]); cout << ", "; Watch(F, mF); cout << ">" << endl;

menu();

//6 Пользователь выбирает исполняемую операцию :

// 6.1 операция пересечения соответствий A и B

// 6.2 операция объединения соответствий A и B

// 6.3 операция разности соответствий A и B

// 6.4 операция разности соответствий B и A

// 6.5 операция симметрической разности соответствий A и B

// 6.6 операция инверсии соответствия A

// 6.7 операция инверсии соответствия B

// 6.8 операция композиции соответствий A и B

// 6.9 операция композиции соответствий B и A

// 6.10 операция нахождения образа соответствия A

// 6.11 операция нахождения прообраза соответствия A

// 6.12 операция нахождения образа соответствия B

// 6.13 операция нахождения прообраза соответствия B

// 6.14 операция сужения соответствия A

// 6.15 операция продолжения соответствия A

// 6.16 операция сужения соответствия B

// 6.17 операция продолжения соответствия B

// 6.18 завершение программы

int Operaziya;

cin >> Operaziya;

system("cls");

cout << "A = <"; WatchMn(X, X[0]); cout << ", "; WatchMn(Y, Y[0]); cout << ", "; Watch(G, mG); cout << ">" << endl;

cout << "B = <"; WatchMn(U, U[0]); cout << ", "; WatchMn(V, V[0]); cout << ", "; Watch(F, mF); cout << ">" << endl << endl;

switch (Operaziya)

{

case 1: // 25 Нахождение пересечения соответствий A и B:

{

cout << "Пересечение соответствий:" << endl;

int mD = mG + mF + 1;

int \*\*D = new int\*[mD];

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

D[i] = new int[3];

D = Peresek(G, F, mG, mF); // 25.1 Нахождение пересечения графиков G и F:

mD = D[0][0];

int \*H = new int[X[0] + U[0]];

H = PeresekMn(X, U, X[0], U[0]); // 25.2 Нахождение пересечения множеств X и U:

int \*J = new int[Y[0] + V[0]];

J = PeresekMn(Y, V, Y[0], V[0]); // 25.3 Нахождение пересечения множеств Y и V:

cout << "<"; WatchMn(H, H[0]); cout << ", "; WatchMn(J, J[0]); cout << ", "; Watch(D, mD); cout << ">" << endl; // 25.5 Выводим соответствие < H, J, C > на экран.

system("pause");

}

break;

case 2: // 26 Нахождение объединения соответствий A и B:

{

cout << "Объединение графиков :" << endl;

int mC = mG + mF + 1;

int \*\*C = new int\*[mC];

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

C[i] = new int[3];

C = InOne(G, F, mG, mF); // 26.1 Нахождение объединения графиков G и F:

mC = C[0][0];

int \*K = new int[X[0] + U[0]];

K = InOneMn(X, U, X[0], U[0]); // 26.2 Нахождение объединения множеств X и U:

int \*L = new int[Y[0] + V[0]];

L = InOneMn(Y, V, Y[0], V[0]); // 26.3 Нахождение объединения множеств Y и V:

cout << "<"; WatchMn(K, K[0]); cout << ", "; WatchMn(L, L[0]); cout << ", "; Watch(C, mC); cout << ">" << endl; // 26.5 Выводим на экран соответствие <K,L, D >.

system("pause");

}

break;

case 3: // 27 Нахождение разности соответствий A и B:

{

cout << "Разность A и В :" << endl;

int mR = mG + 1;

int \*\*R = new int\*[mR];

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

R[i] = new int[3];

R = Pa3HocTb(G, F, mG, mF); // 27.1 Нахождение разности графиков G и F:

mR = R[0][0];

int \*T = new int[X[0] + 1];

T = Pa3HocTbMn(X, U, X[0], U[0]); // 27.2 Нахождение разности множеств X и U:

int \*Q = new int[Y[0] + 1];

Q = Pa3HocTbMn(Y, V, Y[0], V[0]); // 27.3 Нахождение разности множеств Y и V:

cout << "<"; WatchMn(T, T[0]); cout << ", "; WatchMn(Q, Q[0]); cout << ", "; Watch(R, mR); cout << ">" << endl; // 27.5 Выводим соответствие < T,Q, R> на экран.

system("pause");

}

break;

case 4: // 28 Нахождение разности соответствий B и A:

{

cout << "Разность B и A :" << endl;

int mR = mF + 1;

int \*\*R = new int\*[mR];

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

R[i] = new int[3];

R = Pa3HocTb(F, G, mF, mG); // 28.1 Нахождение разности графиков F и G:

mR = R[0][0];

int \*T = new int[U[0] + 1];

T = Pa3HocTbMn(U, X, U[0], X[0]); // 28.2 Нахождение разности множеств U и X:

int \*Q = new int[V[0] + 1];

Q = Pa3HocTbMn(V, Y, V[0], Y[0]); // 28.3 Нахождение разности множеств V и Y:

cout << "<"; WatchMn(T, T[0]); cout << ", "; WatchMn(Q, Q[0]); cout << ", "; Watch(R, mR); cout << ">" << endl;// 28.5 Выводим соответствие < T,Q, R> на экран.

system("pause");

}

break;

case 5: // 29 Нахождение инверсии соответствия A:

{

cout << "Инверсия А :" << endl;

int mI = mG;

int \*\*I = new int\*[mI];

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

I[i] = new int[3];

I = Inversion(G, mG); // 29.1 Нахождение инверсии графика G:

mI = I[0][0];

cout << "<"; WatchMn(Y, Y[0]); cout << ", "; WatchMn(X, X[0]); cout << ", "; Watch(I, mI); cout << ">" << endl; // 29.5 Выводим соответствие <Y,X,I> на экран.

system("pause");

}

break;

case 6: // 30 Нахождение инверсии соответствия B:

{

cout << "Инверсия B :" << endl;

int mI = mF;

int \*\*I = new int\*[mI];

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

I[i] = new int[3];

I = Inversion(F, mF); // 30.1 Нахождение инверсии графика F:

mI = I[0][0];

cout << "<"; WatchMn(V, V[0]); cout << ", "; WatchMn(U, U[0]); cout << ", "; Watch(I, mI); cout << ">" << endl; // 30.5 Выводим соответствие <V,U,I> на экран.

system("pause");

}

break;

case 7: // 31 Нахождение симметрической разности соответствий A и B:

{

cout << "Симметрическая разность A u B :" << endl;

int mS = mF + mG + 1;

int \*\*S = new int\*[mS];

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

S[i] = new int[3];

S = SimRas(F, G, mF, mG); // 31.1 Нахождение симметрической разности графиков G и F :

mS = S[0][0];

int \*O = new int[X[0] + U[0] + 2];

O = SimRasMn(X, U, X[0], U[0]); // 31.2 Нахождение симметрической разности множеств X и U:

int \*H = new int[Y[0] + V[0] + 2];

H = SimRasMn(Y, V, Y[0], V[0]); // 31.3 Нахождение симметрической разности множеств Y и V:

cout << "<"; WatchMn(O, O[0]); cout << ", "; WatchMn(H, H[0]); cout << ", "; Watch(S, mS); cout << ">" << endl; // 31.5 Выводим соответствие <O,H,C > на экран

system("pause");

}

break;

case 8: // 32 Нахождение композиции соответствий A и B:

{

cout << "Композиция соответствий A и B :" << endl;

int mK = mF\*mG + 1;

int \*\*K = new int\*[mK];

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

K[i] = new int[3];

K = Exposition(G, F, mG, mF); // 32.1 Нахождение композиции графиков G и F :

mK = K[0][0];

cout << "<"; WatchMn(X, X[0]); cout << ", "; WatchMn(V, V[0]); cout << ", "; Watch(K, mK); cout << ">" << endl;// 32.5 Выводим соответствие <X,V,К>на экран.

system("pause");

}

break;

case 9: // 33 Нахождение композиции соответствий B и A:

{

cout << "Композиция соответствий B и A :" << endl;

int mK = mF\*mG + 1;

int \*\*K = new int\*[mK];

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

K[i] = new int[3];

K = Exposition(F, G, mF, mG);

mK = K[0][0];

cout << "<"; WatchMn(U, U[0]); cout << ", "; WatchMn(Y, Y[0]); cout << ", "; Watch(K, mK); cout << ">" << endl;// 33.5 Выводим соответствие <U,Y,К> на экран.

system("pause");

}

break;

case 10:// 34 Нахождение образа соответствия A:

{

int mR = (Y[0] \* mG / 2);

int \*R = new int[mR]; // 34.3 Создаем пустое множество R.

R = Obraz(G, mG); // 34 Нахождение образа соответствия A :

mR = R[0];

system("cls");

cout << "Образ A :" << endl;

WatchMn(R, R[0]); cout << endl; // 34.10 Выводим множество R на экран.

system("pause");

}

break;

case 11:// 35 Нахождение прообраза соответствия A:

{

int mZ = (Y[0] \* mG);

int \*Z = new int[mZ]; // 35.3 Создаём пустое множество Z.

Z = ProObraz(G, mG); // 35 Нахождение прообраза соответствия A:

mZ = Z[0];

system("cls");

cout << "ПРОобраз A :" << endl;

WatchMn(Z, Z[0]); cout << endl; // 35.11 Выводим множество Z на экран.

system("pause");

}

break;

case 12:// 36 Нахождение образа соответствия B:

{

int mR = (Y[0] \* mG / 2);

int \*R = new int[mR]; // 36.3 Создаем пустое множество R.

R = Obraz(F, mF); // 36 Нахождение образа соответствия B:

mR = R[0];

system("cls");

cout << "Образ B :" << endl;

WatchMn(R, R[0]); cout << endl;

system("pause");

}

break;

case 13:// 37 Нахождение прообраза соответствия B:

{

int mZ = (U[0] \* mF);

int \*Z = new int[mZ]; // 37.3 Создаём пустое множество Z.

Z = ProObraz(F, mF);

mZ = Z[0]; // 37 Нахождение прообраза соответствия B

system("cls");

cout << "ПРОобраз B :" << endl;

WatchMn(Z, Z[0]); cout << endl; // 37.11 Выводим множество Р на экран.

system("pause");

}

break;

case 14:// 38 Нахождение сужения соответствия A:

{

int mE = mG + Y[0] \* Y[0];

int \*\*E = new int\*[mE];

for (int i = 1; i <= mE; i++)

E[i] = new int[3];

E = Suzko(Y, G, mG); // 38 Нахождение сужения соответствия A:

mE = E[0][0];

system("cls");

cout << "Cужение А :" << endl;

cout << "<"; WatchMn(X, X[0]); cout << ", "; WatchMn(Y, Y[0]); cout << ", "; Watch(E, mE); cout << ">" << endl;// 38.6 Выводим соответствие <X,Y,D> на экран.

system("pause");

}

break;

case 15:// 39 Нахождение продолжения соответствия A:

{

cout << "Продолжение А:\n";

int mJ = mG + 2;

int \*\*J = new int\*[mJ]; // 39.5 Создаем пустой график J.

for (int i = 1; i <= mJ; i++)

J[i] = new int[3];

J = Continue(G, mG); // 39 Нахождение продолжения соответствия A:

mJ = J[0][0];

system("cls");

cout << "Продолжение А:\n";

cout << "<"; WatchMn(X, X[0]); cout << ", "; WatchMn(Y, Y[0]); cout << ", "; Watch(J, mJ); cout << ">" << endl;// 39.9 Выводим соответствие <T,Q, J> на экран.

system("pause");

}

break;

case 16:// 40 Нахождение сужения соответствия B:

{

int mE = mF + V[0] \* V[0];

int \*\*E = new int\*[mE];

for (int i = 1; i <= mE; i++)

E[i] = new int[3];

E = Suzko(V, F, mF);// 40 Нахождение сужения соответствия B:

mE = E[0][0];

system("cls");

cout << "Cужение B :" << endl;

cout << "<"; WatchMn(U, U[0]); cout << ", "; WatchMn(V, V[0]); cout << ", "; Watch(E, mE); cout << ">" << endl;// 40.6 Выводим соответствие <U,V,D> на экран

system("pause");

}

break;

case 17:// 41 Нахождение продолжения соответствия B:

{

cout << "Продолжение В\n";

int mJ = mG + 2;

int \*\*J = new int\*[mJ]; // 41.5 Создаем пустой график J.

for (int i = 1; i <= mJ; i++)

J[i] = new int[3];

J = Continue(F, mF); // 41 Нахождение продолжения соответствия B:

mJ = J[0][0];

system("cls");

cout << "Продолжение В:\n";

cout << "<"; WatchMn(U, U[0]); cout << ", "; WatchMn(V, V[0]); cout << ", "; Watch(J, mJ); cout << ">" << endl; // 41.9 Выводим соответствие <T,Q, J> на экран

system("pause");

}

break;

case 0: // 42 Завершаем алгоритм.

return 0;

}

}

}

int\*\* PereChislenie(int mM)

{

int \*\*M;

M = new int \*[mM];

for (int i = 1; i <= mM; i++)

M[i] = new int[3];

if (mM < 1)

{

cout << "Пустое график!" << endl;

return M;

}

cout << "Введите пары графика соответствия:\n";

for (int i = 1; i <= mM; i++)

{

cout << "Пара №" << i << endl;

for (int j = 1; j < 3; j++)

{

cin >> M[i][j];

}

}

cout << endl;

return M;

}

int\* Z(int left, int right)

{

int \*Spisok = new int[50];

int j = 0;

for (int i = left; i <= right; i++)

{

j++;

Spisok[j] = i;

}

Spisok[0] = j;

return Spisok;

}

void Watch(int \*\*M, int mM)

{

cout << "{";

for (int i = 1; i <= mM; i++)

{

cout << "<";

for (int j = 1; j < 3; j++)

{

cout << M[i][j];

if (j == 1)

cout << ", ";

}

cout << ">";

if (i < mM)

cout << ", ";

}

cout << "}";;

}

void WatchMn(int \*M, int mM)

{

cout << "{";

for (int i = 1; i <= mM; i++)

{

cout << M[i];

if (i < mM)

cout << ", ";

}

cout << "}";

}

void KonZap()

{

cout << "Cоответствия заполнены." << endl;

system("pause");

system("cla");

}

void menu()

{

cout << endl << "Выберите операцию:" << endl;

cout << " 1. Пересечение соответствий.\n 2. Объединение соответствий.\n 3. Разность A и В.\n 4. Разность B и A.\n 5. Инверсия А.\n";

cout << " 6. Инверсия В.\n 7. Симметрическая разность A u B.\n 8. Композиция соответствий A и B.\n 9. Композиция соответствий B и A.\n";

cout << " 10. Образ А.\n 11. Прообраз А.\n 12. Образ В.\n 13. Прообраз В.\n 14. Cужение А.\n 15. Продолжение А.\n 16. Сужение В.\n";

cout << " 17. Продолжение В.\n 0. Выход.\n";

cout << "-->";

}

int\*\* InOne(int \*\*A, int \*\*B, int mA, int mB)

{

int mC = mA + mB + 1;

int i, j;

int \*\*C = new int\*[mC]; // 26.1.1 Создаём пустой график D

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

C[i] = new int[3];

if (mB == 0 && mA == 0) // 26.1.4 Если числа n = 0 и m = 0 одновременно , то график D - пустой график.

return C;

if (mA == 0) // 26.1.2 Если число n = 0, тогда добавляем элементы графика F в график D.

{

for (i = 1; i <= mB; i++)

for (j = 1; j < 3; j++)

{

C[i][j] = B[i][j];

}

C[0][0] = mB;

return C;

}

if (mB == 0) // 26.1.3 Если число m = 0, тогда добавляем элементы графика G в график D

{

for (i = 1; i <= mA; i++)

for (j = 1; j < 3; j++)

{

C[i][j] = A[i][j];

}

C[0][0] = mA;

return C;

}

for (i = 1; i <= mA; i++)

for (j = 1; j < 3; j++)

{

C[i][j] = A[i][j];

}

for (int b = 1; b <= mB; b++)

{

for (int a = 1; a <= mA; a++)

{

if (B[b][1] == A[a][1] && B[b][2] == A[a][2]) // 26.1.7 Если первая компонента i-й пары графика G равна первой компоненте j-й пары графика F

// 26.1.7.1 Вторая компонента i - й пары графика G равна второй компоненте j - й пары графика F, переходим к пункту 26.1.12.

break;

if (B[b][1] != A[a][1] || B[b][2] != A[a][2]) // 26.1.8 Если первая компонента i-й пары равна первой компоненте одной из пар графика D и вторая компонента i-й пары равна второй компоненте той же пары графика D, то переходим к пункту 26.1.12.

{

if (a == (mA))

{

C[i][1] = B[b][1]; // 26.1.9 Добавляем i-ю пару в график D.

C[i][2] = B[b][2];

i++;

}

else continue;

}

}

}

C[0][0] = i - 1;

return C;

}

int\* InOneMn(int \*A, int \*B, int mA, int mB)

{

int mO = mA + mB + 2, i = 0;

int \*C = new int[mO]; // 26.2.1 Создаём пустое множество К

for (i = 1; i<mA; i++)

C[i] = A[i];

C[i] = A[i]; // 26.2.11 Добавляем j-й элемент во множество K.

for (int b = 1; b <= mB; b++)

for (int a = 1; a <= mA; a++)

{

if (B[b] == A[a]) // 26.2.7 Если i-й элемент множества X и j-й элемент множества U равны, переходим к пункту 26.2.12.

break;

if (B[b] != A[a]) //

{

if (a == (mA))

{

i++;

C[i] = B[b]; // 26.2.9 Добавляем i-й элемент во множество K.

}

else continue;

}

}

C[0] = i;

return C;

}

int\*\* Peresek(int \*\*A, int \*\*B, int mA, int mB)

{

int mC = mA + mB + 1; // 25.1.1 Создаём пустой график С.

int \*\*C = new int\*[mC];

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

C[i] = new int[3];

int i = 0;

if (mA == 0 || mB == 0) // 25.1.2 Если число n = 0, тогда график C - пустой график.

return C; // 25.1.3 Если число m = 0, тогда пересечение C - пустой график.

for (int b = 1; b <= mB; b++)

for (int a = 1; a <= mA; a++)

if (B[b][1] == A[a][1] && B[b][2] == A[a][2]) // 25.1.6 Если первая компонента i-й пары графика G не равна первой компоненте j-й пары графика F, переходим к пункту 25.1.9.

// 25.1.7 Если вторая компонента i - й пары графика G не равна второй компоненте j - й пары графика F, переходим к пункту 25.1.9.

{

i++;

C[i][1] = B[b][1]; // 25.1.8 Добавляем i-ю пару графика G в график C.

C[i][2] = B[b][2];

}

C[0][0] = i;

return C;

}

int\* PeresekMn(int \*A, int\*B, int mA, int mB)

{

int \*H = new int[mB + mA]; // 25.2.1 Создаём пустое множество H.

int i = 0;

if (mA == 0 || mB == 0) // 25.2.2 Если число n = 0, тогда множество Н - пустое множество.

// 25.2.3 Если число m = 0, тогда пересечение H - пустое множество.

return H; // 25.2.3.1 Переходим к пункту 25.2.11.

for (int b = 1; b <= mB; b++)

for (int a = 1; a <= mA; a++)

if (B[b] == A[a]) // 25.2.6 Если i-й элемент множества X и j-й элемент множества U равны, то добавляем этот элемент во множество H.

{

i++;

H[i] = B[b];

}

H[0] = i;

return H;

}

int\*\* Pa3HocTb(int \*\*U, int \*\*W, int mU, int mW)

{

int mR = mU + 1;

int \*\*R = new int\*[mR]; // 27.1.1 Создаём пустой график R.

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

R[i] = new int[3];

int r = 0;

if (mU == 0) // 27.1.2 Если число n равно нулю, тогда график R - пустой график.

return R; // 27.1.2.1 Переходим к пункту 27.1.12.

if (mW == 0)

{

for (int i = 1; i <= mU; i++) // 27.1.3 Если число m равно нулю, тогда добавляем элементы графика G в график R.

{

R[i][1] = U[i][1];

R[i][2] = U[i][2];

}

R[0][0] = mU;

return R; // 27.1.2.1 Переходим к пункту 27.1.12.

}

for (int i = 1; i <= mU; i++)

for (int j = 1; j <= mW; j++)

{

if (U[i][1] == W[j][1] && U[i][2] == W[j][2]) // 27.1.6 Если первая компонента i-й пары графика G равна первой компоненте j-й пары графика F

// 27.1.6.1 Вторая компонента i - й пары графика G равна второй компоненте j - й пары графика F, переходим к пункту 2.1.10.

break;

if (j == (mW))

{

r++;

R[r][1] = U[i][1]; // 27.1.9 Добавляем i-ю пару в график R.

R[r][2] = U[i][2];

}

}

R[0][0] = r;

return R;

}

int\* Pa3HocTbMn(int \*U, int \*W, int mU, int mW)

{

int \*T = new int[mU + 1]; // 27.2.1 Создаём пустое множество Т.

int r = 0;

for (int i = 1; i <= mU; i++)

for (int j = 1; j <= mW; j++)

{

if (U[i] == W[j]) // 27.2.6 Если i-й элемент множества X и j-й элемент множества U равны, переходим к пункту 27.2.10.

break;

if (j == (mW)) // 27.2.11 Если i меньше n, переходим к пункту 27.2.6.

{

r++;

T[r] = U[i]; // 27.2.9 Добавляем i-й элемент во множество T.

}

}

T[0] = r;

return T;

}

int\*\* SimRas(int \*\*A, int \*\*B, int mA, int mB)

{

int mS = mA \* mB + 2; // 31.1.1 Создаём пустой график С.

int \*\*S = new int\*[mS];

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

S[i] = new int[3];

int s = 0, o = 0;

if (mA == 0 && mB == 0) // 31.1.2 Если числа n и m одновременно равны нулю, тогда график S - пустой график.

return S; // Переходим к пункту 31.1.16.

if (mA == 0)

{

for (int i = 1; i <= mB; i++)

{

S[i][1] = B[i][1];

S[i][2] = B[i][2];

}

S[0][0] = mB;

return S;

}

if (mB == 0)

{

for (int i = 1; i <= mA; i++)

{

S[i][1] = A[i][1];

S[i][2] = A[i][2];

}

S[0][0] = mA;

return S;

}

for (int i = 1; i <= mA; i++)

for (int j = 1; j <= mB; j++)

{

o = 0;

if (A[i][1] == B[j][1] && A[i][2] == B[j][2]) // 31.1.5 Сравниваем i-ю пару графика G и j-ю пару графика F.

continue;

for (int l = 1; l <= s; l++)

{

if (S[l][1] == A[i][1] && S[l][2] == A[i][2])

{

o = 1;

break;

}

}

for (int v = 1; v <= mB; v++)

{

if (A[i][1] == B[v][1] && A[i][2] == B[v][2]) //// 31.1.5 Сравниваем i-ю пару графика G и j-ю пару графика F.

{

o = 1;

break;

}

}

if (o == 1)

continue;

s++;

S[s][1] = A[i][1]; // 31.1.11 Добавляем в график разности K пары графика G.

S[s][2] = A[i][2];

}

for (int i = 1; i <= mB; i++)

for (int j = 1; j <= mA; j++)

{

o = 0;

if (B[i][1] == A[j][1] && B[i][2] == A[j][2]) // 31.1.14 Сравниваем k-ю пару графика K и j-ю пару графика С.

continue;

for (int l = 1; l <= s; l++)

{

if (S[l][1] == B[i][1] && S[l][2] == B[i][2]) //Если первая компонента k-й пары графика K и первая компонента j-й пары графика С равны, то переходим к пункту 31.1.14.2

{

o = 1;

break;

}

}

for (int v = 1; v <= mA; v++)

{

if (B[i][1] == A[v][1] && B[i][2] == A[v][2]) // 31.1.14.2 Если вторая компонента k-й пары графика K и вторая компонента j-й пары графика С равны, то удаляем эту пару из графика K.

{

o = 1;

break;

}

}

if (o == 1)

continue;

s++;

S[s][1] = B[i][1]; // 31.1.20 Добавляем в график разности M пары графика B

S[s][2] = B[i][2];

}

S[0][0] = s;

return S;

}

int\* SimRasMn(int \*A, int \*B, int mA, int mB)

{

int \*S = new int[mA + mB + 2]; // 31.2.1 Создаём пустое множество О

int s = 0, o = 0;

for (int i = 1; i <= mA; i++)

for (int j = 1; j <= mB; j++)

{

o = 0;

if (A[i] == B[j]) // 31.2.5 Если i-й элемент множества X и j-й элемент множества U равны, то переходим к пункту 31.2.13.

continue;

for (int l = 1; l <= s; l++)

{

if (S[l] == A[i]) // 31.2.6 Если i-й элемент равен одному из элементов множества O, то переходим к пункту 31.2.12.

{

o = 1;

break;

}

}

for (int v = 1; v <= mB; v++)

{

if (A[i] == B[v]) // 31.2.7 Если i-й элемент равен одному из элементов множества U, то переходим к пункту 31.2.12.

{

o = 1;

break;

}

}

if (o == 1)

continue;

s++;

S[s] = A[i]; // 31.2.8 Добавляем i-й элемент во множество O.

}

for (int i = 1; i <= mA; i++)

for (int j = 1; j <= mB; j++)

{

o = 0;

for (int l = 1; l <= s; l++)

{

if (S[l] == B[j]) //// 31.2.9 Если j-й элемент равен одному из элементов множества O, то переходим к пункту 31.2.12.

{

o = 1;

break;

}

}

for (int v = 1; v <= mA; v++)

{

if (A[v] == B[j]) // 31.2.10 Если j-й элемент равен одному из элементов множества X, то переходим к пункту 31.2.12.

{

o = 1;

break;

}

}

if (o == 1)

continue;

s++;

S[s] = B[j]; // 31.2.11 Добавляем j-й элемент во множество O.

}

S[0] = s;

return S;

}

int\*\* Inversion(int \*\*M, int mM)

{

int \*\*I = new int\*[mM]; // 29.1.1 Создаём пустой график I.

int i = 0;

for (int l = 0; l <= mM; l++)

I[l] = new int[3];

if (mM == 0) // 29.1.2 Если число n равно нулю, тогда график I – пустой график.

return I; // Переходим к пункту 29.1.8.

for (i = 1; i <= mM; i++)

{

I[i][1] = M[i][2]; // 29.1.4.1 Добавляем на место первой компоненты вторую компоненту i-й пары графика G.

I[i][2] = M[i][1]; // 29.1.4.2 Добавляем на место второй компоненты первую компоненту i-й пары графика G.

}

I[0][0] = mM;

return I;

}

int\*\* Exposition(int \*\*A, int \*\*B, int mA, int mB)

{

int mO = mA \* mB + 1;

int \*\*K = new int\*[mO]; // 32.1.1 Создаём пустой график К.

int k = 0;

for (int p = 0; p <= mO; p++)

K[p] = new int[3];

if (mA == 0 || mB == 0) // 32.1.2 Если число n = 0, тогда график К - пустой график.

// 32.1.3 Если число m = 0, тогда график К - пустой график.

return K; // 32.1.2.1 Переходим к пункту 32.1.13.

for (int i = 1; i <= mA; i++)

for (int j = 1; j <= mB; j++)

{

int o = 0;

if (A[i][2] != B[j][1])

continue;

for (int l = 1; l <= k; l++)

if (A[i][1] == K[l][1] && B[j][2] == K[l][2]) // 32.1.6 Если вторая компонента i-й пары графика G не равна первой компоненте j-й пары графика F, переходим к пункту 32.1.9.

{

o = 1;

break;

}

if (o == 1)

continue;

k++;

K[k][1] = A[i][1]; // 32.1.8 Добавляем пару а в график K.

K[k][2] = B[j][2];

o = 0;

}

K[0][0] = k;

return K;

}

int\* Obraz(int \*\*G, int mG)

{

cout << "Введите мощность множества для образа:\t";

int mO;

cin >> mO;

int \*O = new int[mO]; // 34.1 Создаём пустое множество О.

cout << "Введите " << mO << " элемента(ов) множества.";

for (int u = 1; u <= mO; u++) // 34.2 Пользователь добавляет элементы в O.

{

cout << "\nЭлемент №" << u << ": ";

cin >> O[u];

}

int mR = 0;

int \*R = new int[mG\*mO]; // 34.3 Создаем пустое множество R.

for (int i = 1; i <= mO; i++)

for (int j = 1; j <= mG; j++)

if (O[i] == G[j][2]) // 34.6 Сравниваем выбранный элемент O со второй компонентой выбранного элемента G:

{

mR++;

R[mR] = G[j][1]; // 34.6.1.1 Записываем в R первую компоненту выбранного элемента графика G.

}

R[0] = mR;

return R;

}

int\* ProObraz(int \*\*G, int mG)

{

cout << "Введите мощность множества для прообраза:\t";

int mP;

cin >> mP;

int \*P = new int[mP]; // 35.1 Создаём пустое множество Р.

cout << "Введите " << mP << " элемента(ов) множества.";

for (int u = 1; u <= mP; u++) // 35.2 Пользователь добавляет элементы в P.

{

cout << "\nЭлемент №" << u << ": ";

cin >> P[u];

}

int mZ = 0;

int \*Z = new int[mG\*mP];

for (int i = 1; i <= mP; i++)

for (int j = 1; j <= mG; j++)

if (P[i] == G[j][1]) // 35.8 Сравниваем выбранный элемент P с первой компонентой выбранного элемента G:

{

mZ++;

Z[mZ] = G[j][2]; // 35.8.1.1 Записываем в Z вторую компоненту выбранного элемента графика G.

}

Z[0] = mZ;

return Z;

}

int\*\* Suzko(int \*Y, int \*\*G, int mG)

{

cout << "Введите мощность множества для сужения:\t"; // Полтзователь вводит мощность графика D

/\*int mD;

cin >> mD;

int \*\*D = new int\*[mD]; // 38.2.1 Создаём пустой график Е

for (int i = 1; i <= mD; i++)

D[i] = new int[3];

cout << "Введите " << mD << " пар(ы) графика.\n";

for (int u = 1; u <= mD; u++) // 38.3 Пользователь добавляет два произвольных кортежа в график J.

{

cout << "Пара №" << u << ":\n";

cin >> D[u][1];

cin >> D[u][2];

}\*/

int mS;

cin >> mS;

int \*S = new int[mS]; // 38.1 Создаем пустое множество S.

cout << "Введите " << mS << " элемента(ов) множества.";

for (int u = 1; u <= mS; u++) // 38.2 Пользователь добавляет элементы в S.

{

cout << "\nЭлемент №" << u << ": ";

cin >> S[u];

}

int mR = 0;

int mD = Y[0] \* mS;

int \*\*D = new int \*[mD];

for (int i = 1; i <= mD; i++)

D[i] = new int[3];

D = Dekart(S, Y, mS, Y[0], mD); // 38.3 Находим декартово произведение множеств S и Y:

mD = D[0][0];

int mE = mG + mD;

int \*\*E = new int\*[mE]; // 38.4.1 Создаём пустой график Е

for (int i = 1; i <= mE; i++)

E[i] = new int[3];

E = Peresek(G, D, mG, mD); // 38.4 Находим пересечение графиков G и D:

return E;

}

int\*\* Dekart(int \*A, int \*B, int mA, int mB, int mD)

{

int \*\*D = new int \*[mD];

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

D[i] = new int[3];

int o = 0; int d = 0;

for (int i = 1; i <= mA; i++)

for (int j = 1; j <= mB; j++)

{

o = 0;

for (int k = 1; k < d; k++)

{

if (A[i] == D[k][1] && B[j] == D[k][2])

{

o = 1;

break;

}

}

if (o == 1)

continue;

d++;

D[d][1] = A[i];

D[d][2] = B[j];

}

D[0][0] = d;

return D;

}

int\*\* Continue(int \*\*G, int mG)

{

cout << "Введите количество пар, для добавления в график:\t";

int k; cin >> k; // Пользователь вводит к элементов, для добавления в график

int mJ = mG + k;

int \*\*J = new int\*[mJ]; // 39.1 Создаем пустой график J.

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

J[i] = new int[3];

for (int i = 1; i <= mG; i++) // 39.2 Добавляем в J все элементы графика G.

{

J[i][1] = G[i][1];

J[i][2] = G[i][2];

}

cout << "Введите" << k <<" пар(ы) для для добавления в график:\n";

for (int u = mG + 1; u <= mJ; u++) // 39.4 Пользователь последовательно вводит k пар .

{

cout << "Пара №" << u << ":\n";

cin >> J[u][1]; //39.5 Добавляем введённые пользователем пары в график J.

cin >> J[u][2];

}

J[0][0] = mJ;

return J;

}