**Пермский национальный исследовательский политехнический университет**

Кафедра “Информационные технологии и автоматизированные системы”

**Творческие задания**

«По дисциплине «Основы алгоритмизации и программирования»

**Тема:**

Разработка калькулятора и задача коммивояжера

Выполнила:

студентка группы ИВТ-19-1б

Еремеева Полина Алексеевна

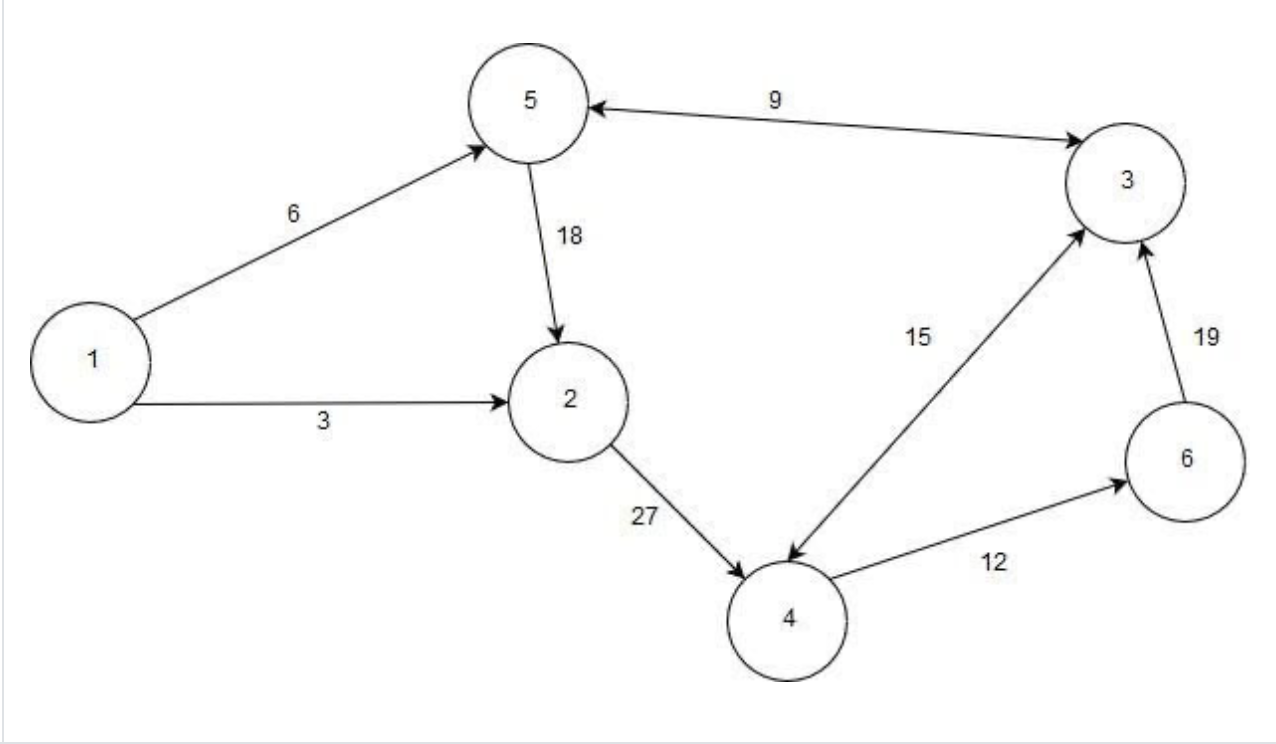
Проверила:

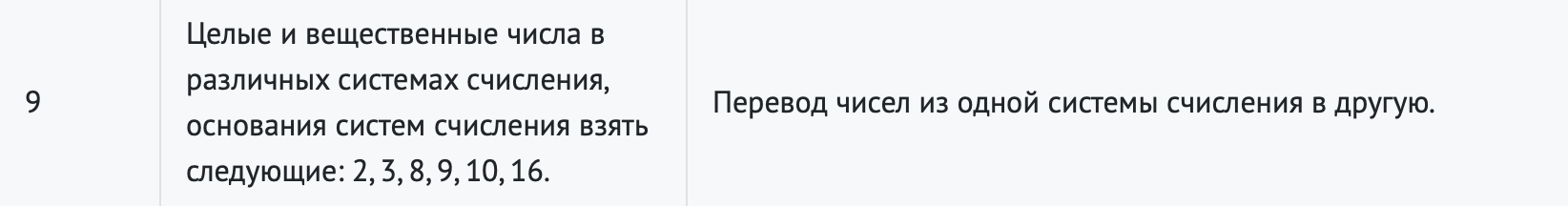
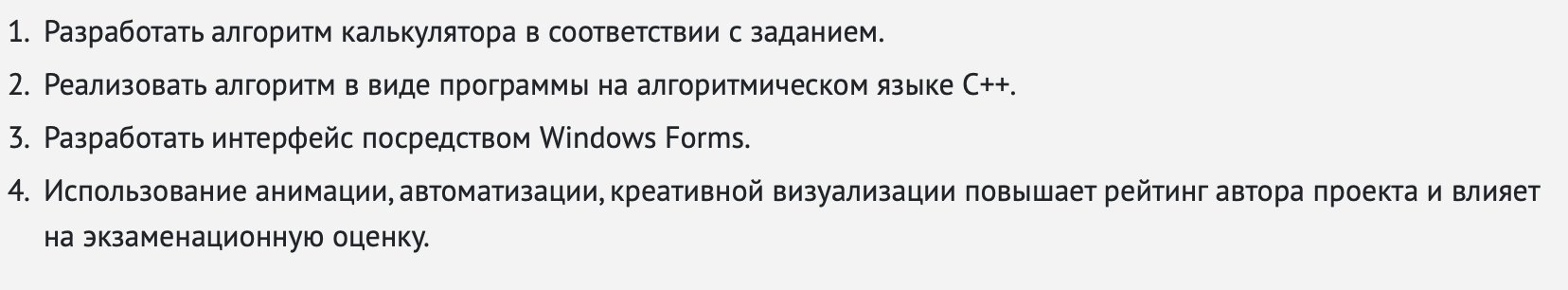
доцент кафедры “ИТАС”

Полякова О.А.

Пермь, 2020

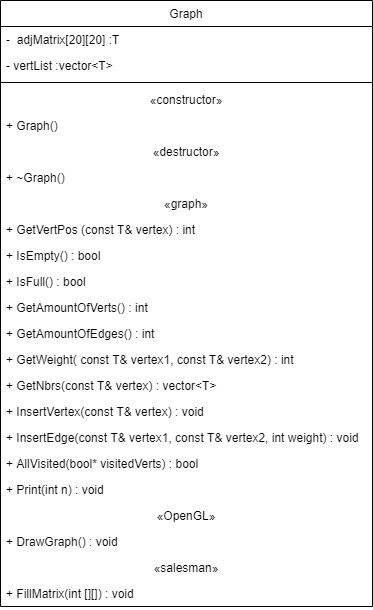
**Постановка задачи**

1) Написать алгоритм решения задачи коммивояжёра для своего варианта из лабораторной работы №31.

2)

**Код**

Задача коммивояжера



mainlab.cpp

#pragma once

#include <iostream>

#include <vector>

#include "Graph.h"

#include <glut.h>

#include <string>

using namespace std;

int main(int argc, char\* argv[])

{

setlocale(LC\_ALL, "ru");

glutInit(&argc, argv);

graph = makeGraph();

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGBA);

glutInitWindowSize(900, 720);

glutCreateWindow("Граф");

WinW = glutGet(GLUT\_WINDOW\_WIDTH);

WinH = glutGet(GLUT\_WINDOW\_HEIGHT);

glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutMainLoop();

return 0;

}

Graph.h

#pragma once

#include <iostream>

#include <vector>

#include "glut.h"

#include <string>

using namespace std;

extern int maxSize;

extern const int za;

extern int WinW, WinH, R;

template <typename T>

class Graph

{

public:

Graph();

~Graph();

int GetVertPos(const T& vertex);

bool IsEmpty();

bool IsFull();

int GetAmountOfVerts();

int GetAmountOfEdges();

int GetWeight(const T& vertex1, const T& vertex2);

vector<T> GetNbrs(const T& vertex);

void InsertVertex(const T& vertex);

void InsertEdge(const T& vertex1, const T& vertex2, int weight);

void FillLabels(T& startVertex);

int Dijsktra(T& startVertex);

bool AllVisited(bool\* visitedVerts);

void Print();

void DrawGraph();

void fillmatrix(int mat[6][6]);

private:

T adjMatrix[20][20];

vector<T> vertList;

vector<int> labelList;

};

extern Graph <int> graph;

template <typename T>

void Graph<T>::fillmatrix(int mat[6][6])

{

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

{

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

{

if (i == j)

{

mat[i][j] = 0;

}

else

{

mat[i][j] = adjMatrix[i][j];

}

}

}

}

template <typename T>

Graph<T>::Graph()

{

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

{

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

{

this->adjMatrix[i][j] = 0;

}

}

}

template <typename T>

Graph<T>::~Graph()

{

}

template <typename T>

int Graph<T>::GetVertPos(const T& vertex)

{

for (int i = 0; i < this->vertList.size(); ++i)

{

if (this->vertList[i] == vertex) return i;

}

return -1;

}

template <typename T>

bool Graph<T>::IsEmpty()

{

if (this->vertList.size() != 0) return false;

else return true;

}

template <typename T>

bool Graph<T>::IsFull()

{

return (vertList.size() == maxSize);

}

template <typename T>

int Graph<T>::GetAmountOfVerts()

{

return this->vertList.size();

}

template <typename T>

int Graph<T>::GetAmountOfEdges()

{

int amount = 0;

if (!this->IsEmpty())

{

for (int i = 0, vertListSize = this->vertList.size(); i < vertListSize; ++i)

{

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

{

if (this->adjMatrix[i][j] == this->adjMatrix[j][i] && this->adjMatrix[i][j] != 0) amount += 1;

}

}

return (amount / 2);

}

else return 0;

}

template <typename T>

int Graph<T>::GetWeight(const T& vertex1, const T& vertex2)

{

if (!this->IsEmpty())

{

int vertPos1 = GetVertPos(vertex1);

int vertPos2 = GetVertPos(vertex2);

return adjMatrix[vertPos1][vertPos2];

}

else return 0;

}

template <typename T>

vector<T> Graph<T>::GetNbrs(const T& vertex)

{

vector<T> nbrsList;

int vertPos = this->GetVertPos(vertex);

if (vertPos != (-1))

{

for (int i = 0, vertListSize = this->vertList.size(); i < vertListSize; ++i)

{

if (this->adjMatrix[vertPos][i] != 0 && this->adjMatrix[i][vertPos] != 0) nbrsList.push\_back(this->vertList[i]);

}

}

return nbrsList;

}

template <typename T>

void Graph<T>::InsertVertex(const T& vertex)

{

if (!this->IsFull()) this->vertList.push\_back(vertex);

else

{

cout << "Граф заполнен. Невохможно добавить новую вершину" << endl;

return;

}

}

template <typename T>

void Graph<T>::InsertEdge(const T& vertex1, const T& vertex2, int weight)

{

if (this->GetVertPos(vertex1) != (-1) && this->GetVertPos(vertex2) != (-1))

{

int vertPos1 = GetVertPos(vertex1);

int vertPos2 = GetVertPos(vertex2);

if (this->adjMatrix[vertPos1][vertPos2] != 0 && this->adjMatrix[vertPos2][vertPos1] != 0)

{

cout << "Ребро между этими вершинами уже есть" << endl;

return;

}

else

{

this->adjMatrix[vertPos1][vertPos2] = weight;

this->adjMatrix[vertPos2][vertPos1] = weight;

}

}

else

{

cout << "Одной из вершин (или обеих)нет в графе" << endl;

return;

}

}

template <typename T>

void Graph<T>::FillLabels(T& startVertex)

{

for (int i = 0, size = vertList.size(); i < size; ++i)

{

labelList.push\_back(1000000);

}

int pos = GetVertPos(startVertex);

labelList[pos] = 0;

}

template <typename T>

bool Graph<T>::AllVisited(bool\* visitedVerts)

{

bool flag = true;

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

{

if (visitedVerts[i] != true) flag = false;

}

return flag;

}

template <typename T>

int Graph<T>::Dijsktra(T& startVertex)

{

FillLabels(startVertex);

bool\* visitedVerts = new bool[vertList.size()];

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

{

visitedVerts[i] = false;

}

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

{

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

{

if (adjMatrix[i][j] < 0) return -1;

if (GetVertPos(startVertex) == -1) return -2;

}

}

T curSrc;

int counter = 0;

int minLabel = 1000000;

vector<T>neighbors = GetNbrs(startVertex);

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

{

int startLabel = labelList[GetVertPos(startVertex)];

int weight = GetWeight(startVertex, neighbors[i]);

int nIndex = GetVertPos(neighbors[i]);

int nextLabel = labelList[nIndex];

if (startLabel + weight < nextLabel) labelList[nIndex] = startLabel + weight;

if (labelList[nIndex] < minLabel) minLabel = labelList[nIndex];

}

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

{

if (labelList[GetVertPos(neighbors[i])] != 1000000) counter += 1;

if (counter == neighbors.size()) visitedVerts[GetVertPos(startVertex)] = true;

}

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

{

if (labelList[GetVertPos(neighbors[i])] == minLabel) curSrc = neighbors[i];

}

while (!AllVisited(visitedVerts))

{

neighbors = GetNbrs(curSrc);

int count = 0;

int minLabel = 1000000;

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

{

int curLabel = labelList[GetVertPos(curSrc)];

int weight = GetWeight(curSrc, neighbors[i]);

int nIndex = GetVertPos(neighbors[i]);

int nextLabel = labelList[nIndex];

if (curLabel + weight < nextLabel) labelList[nIndex] = curLabel + weight;

if (labelList[nIndex] < minLabel && visitedVerts[nIndex] != true) minLabel = labelList[nIndex];

count += 1;

}

if (count == neighbors.size()) visitedVerts[GetVertPos(curSrc)] = true;

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

{

if (labelList[GetVertPos(neighbors[i])] == minLabel && visitedVerts[GetVertPos(neighbors[i])] != true) curSrc = neighbors[i];

}

}

for (int i = 0; i < GetVertPos(startVertex); ++i)

{

cout << "Кратчайшее расстояние от вершины:" << startVertex << " "

<< "До вершины:" << vertList[i] << " " << "равно" << "-" << labelList[GetVertPos(vertList[i])] << endl;

}

for (int i = GetVertPos(startVertex) + 1; i < vertList.size(); ++i)

{

cout << "Кратчайшее расстояние от вершины:" << startVertex << " "

<< "До вершины:" << vertList[i] << " " << "равно" << "-" << labelList[GetVertPos(vertList[i])] << endl;

}

}

template <typename T>

void Graph<T>::Print()

{

if (!this->IsEmpty())

{

cout << "Матрица смежности графа:" << endl;

cout << " " << " ";

for (int i = 0, vertListSize = this->vertList.size(); i < vertListSize; ++i)

{

cout << " " << this->vertList[i] << " ";

}

cout << endl;

for (int i = 0, vertListSize = this->vertList.size(); i < vertListSize; ++i)

{

cout << this->vertList[i] << " ";

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

{

cout << " " << this->adjMatrix[i][j] << " ";

}

cout << endl;

}

}

else cout << "Граф пуст" << endl;

}

Graph<int> makeGraph();

extern struct vertCoord;

struct vertCoord

{

int x, y;

};

extern vertCoord vertC[20];

void setCoord(int i, int n);

void drawCircle(int x, int y, int R);

void drawText(int num, int x1, int y1);

void drawVertex(int n);

void drawLine(int text, int x0, int y0, int x1, int y1);

void reshape(int w, int h);

void display();

template <typename T>

void Graph<T>::DrawGraph()

{

int n = vertList.size();

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

{

setCoord(i, n);

}

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

{

for (int j = i + 1; j < n; ++j)

{

int a = adjMatrix[i][j];

if (a != 0)

{

drawLine(a, vertC[i].x, vertC[i].y, vertC[j].x, vertC[j].y);

}

}

}

drawVertex(n);

}

Graph.cpp

#pragma once

#include "Graph.h"

#include <vector>

#include <iostream>

#include <string>

#include "glut.h"

#include <cmath>

#include "Header.h"

using namespace std;

const int za = 6;

int maxSize = 20;

Graph <int> graph;

int WinW, WinH, R;

extern int mas[6][6];

vertCoord vertC[20];

Graph<int> makeGraph()

{

Graph<int>graph;

int amountofVerts, amountofEdges, vertex, sourceVertex, targetVertex, edgeWeight;

cout << "Введите количество вершин графа:"; cin >> amountofVerts;

cout << "Введите количество ребер графа:"; cin >> amountofEdges;

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

{

cout << "Вершина:"; cin >> vertex;

int\* vertPtr = &vertex;

graph.InsertVertex(\*vertPtr);

cout << endl;

}

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

{

cout << "Исходная вершина:"; cin >> sourceVertex;

int\* sourceVertPtr = &sourceVertex;

cout << "Конечная вершина:"; cin >> targetVertex;

int\* targetVertPtr = &targetVertex;

cout << "Вес ребра:"; cin >> edgeWeight;

graph.InsertEdge(\*sourceVertPtr, \*targetVertPtr, edgeWeight);

}

cout << endl;

graph.Print();

cout << "Стартовая вершина:"; cin >> sourceVertex;

int\* sourceVertPtr = &sourceVertex;

/\*graph.Dijsktra(\*sourceVertPtr);\*/

graph.fillmatrix(mas);

\_input();

run(1, za);

\_output(1, za);

return graph;

}

void setCoord(int i, int n)

{

int R\_;

int x0 = WinW / 2;

int y0 = WinH / 2;

if (WinW > WinH)

{

R = 5 \* (WinH / 13) / n;

R\_ = WinH / 2 - R - 10;

}

else

{

R = 5 \* (WinW / 13) / n;

R\_ = WinW / 2 - R - 10;

}

float theta = 2.0f \* 3.1415926f \* float(i) / float(n);

float y1 = R\_ \* cos(theta) + y0;

float x1 = R\_ \* sin(theta) + x0;

vertC[i].x = x1;

vertC[i].y = y1;

}

void drawCircle(int x, int y, int R)

{

glColor3f(1.0, 1.0, 1.0);

float x1, y1;

glBegin(GL\_POLYGON);

for (int i = 0; i < 360; ++i)

{

float theta = 2.0f \* 3.1415926f \* float(i) / float(360);

y1 = R \* cos(theta) + y;

x1 = R \* sin(theta) + x;

glVertex2f(x1, y1);

}

glEnd();

glColor3f(0.0f, 0.0f, 0.0f);

float x2, y2;

glBegin(GL\_LINE\_LOOP);

for (int i = 0; i < 360; ++i)

{

float theta = 2.0f \* 3.1415926f \* float(i) / float(360);

y2 = R \* cos(theta) + y;

x2 = R \* sin(theta) + x;

glVertex2f(x2, y2);

}

glEnd();

}

void drawText(int num, int x1, int y1)

{

GLvoid\* font = GLUT\_BITMAP\_HELVETICA\_18;

string s = to\_string(num);

glRasterPos2i(x1 - 5, y1 - 5);

for (int j = 0; j < s.length(); ++j)

{

glutBitmapCharacter(font, s[j]);

}

}

void drawVertex(int n)

{

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

{

drawCircle(vertC[i].x, vertC[i].y, R);

drawText(i + 1, vertC[i].x, vertC[i].y);

}

}

void drawLine(int text, int x0, int y0, int x1, int y1)

{

glColor3f(0.0f, 0.0f, 0.0f);

glBegin(GL\_LINES);

glVertex2i(x0, y0);

glVertex2i(x1, y1);

glEnd();

drawText(text, (x0 + x1) / 2 + 10, (y0 + y1) / 2 + 10);

}

void reshape(int w, int h)

{

WinW = w;

WinH = h;

glViewport(0, 0, (GLsizei)WinW, (GLsizei)WinH);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, (GLdouble)WinW, 0, (GLdouble)WinH);

glutPostRedisplay();

}

void display()

{

glShadeModel(GL\_SMOOTH);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, WinW, 0, WinH);

glViewport(0, 0, WinW, WinH);

glClearColor(1.0, 1.0, 1.0, 1.0);

glClear(GL\_COLOR\_BUFFER\_BIT);

graph.DrawGraph();

glutSwapBuffers();

}

Header.h

#pragma once

#include <iostream>

#include <fstream>

using namespace std;

const int maxn = 100;

int mas[6][6];

int n, i, s, min, c, found;

int a[maxn][maxn];

int m[maxn], minm[maxn];

void \_input()

{

ifstream fin("matrix.txt");

fin >> n;

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

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

fin >> a[i][j];

}

void \_output(int q, int n)

{

if (found)

{

cout << "Вес минимального пути: " << min << endl;

cout << "Путь: ";

int c = 1;

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

{

int j = 1;

while ((j <= n) &&

(minm[j] != c)) j++;

cout << j << "->";

c++;

}

cout << minm[1] << endl;

}

else cout << "Путь не найден";

}

void search(int x, int n)

{

if ((c == n) &&

(a[x][1] != 0) &&

(s + a[x][1] < min))

{

found = 1;

min = s + a[x][1];

for (int i = 1; i <= n; i++)minm[i] = m[i];

}

else

{

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

if ((i != x) &&

(a[x][i] != 0) &&

(m[i] == 0) &&

(s + a[x][i] < min))

{

s += a[x][i];

c++;

m[i] = c;

search(i, n);

m[i] = 0;

c--;

s -= a[x][i];

}

}

}

void run(int q, int n)

{

s = 0;

found = 0;

min = 32767;

for (int i = 1; i <= n; i++) m[i] = 0;

c = 1;

m[q] = c;

search(q, n);

}

Разработка калькулятора

MyForm.сpp

#include <iostream>

#include <cmath>

#include <vector>

#include "MyForm.h"

#include "Convector.h"

**using** **namespace** System;

**using** **namespace** System::Windows::Forms;

[STAThreadAttribute]

**void** main(array<String^>^ args)

{

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(**false**);

À‡·33::MyForm from;

Application::Run(% from);

}

System::Void À‡·33::MyForm::button1\_Click(System::Object^ sender, System::EventArgs^ e)

{

**int** num1, num2;

std::string date;

**bool** f = **true**;

std::vector<**int**> vhod;

textBox1->Text = "";

Convector\_String\_to\_string(textBox4->Text->ToString(), date);

num1 = Convert::ToInt32(textBox2->Text);

num2 = Convert::ToInt32(textBox3->Text);

**if** (num1 > 32){

f = **false**;

}

**if** (num2 < 2){

f = **false**;

}

**if** (num2 > 32){

f = **false**;

}

**while** (date.size() > 0){

**if** ((**int**)date[date.size() - 1] >= 48 && (**int**)date[date.size() - 1] <= 57){

vhod.insert(vhod.begin(), (**int**)date[date.size() - 1] - 48);

}

**else** **if** ((**int**)date[date.size() - 1] >= 65 && (**int**)date[date.size() - 1] <= 90){

vhod.insert(vhod.begin(), (**int**)date[date.size() - 1] - 65 + 10);

}

date.erase(date.size() - 1, 1);

}

**for** (**int** i = 0; i < vhod.size(); i++){

**if** (vhod[i] > num1){

f = **false**;

}

}

**if** (f == **false**){

MessageBox::Show("Error!", "Calculator");

}

**int** sum = 0;

**for** (**int** i = 0; i < vhod.size(); i++){

sum += vhod[vhod.size() - 1 - i] \* pow(num1, i);

}

**int** summ = sum;

std::vector<**int**> v;

**while** (summ > 0){

v.insert(v.begin(), summ % num2);

summ = summ / num2;

}

**int** j = 0;

**if** (f == **true**){

**char** r[10] = "";

**for** (**int** i = 0; i < v.size(); i++){

**if** (v[i] < 10){

textBox1->Text += v[i].ToString();

}

**else**{

r[0] = 97 + v[i] - 10 - 32;

textBox1->Text += gcnew System::String(r);

j++;

}

}

}

**return** System::Void();

}

System::Void À‡·33::MyForm::button2\_Click(System::Object^ sender, System::EventArgs^ e)

{

**return** System::Void();

}

System::Void À‡·33::MyForm::label2\_Click(System::Object^ sender, System::EventArgs^ e)

{

**return** System::Void();

}

System::Void À‡·33::MyForm::label4\_Click(System::Object^ sender, System::EventArgs^ e)

{

**return** System::Void();

}

System::Void À‡·33::MyForm::label3\_Click(System::Object^ sender, System::EventArgs^ e)

{

**return** System::Void();

}

System::Void À‡·33::MyForm::label1\_Click(System::Object^ sender, System::EventArgs^ e)

{

**return** System::Void();

}

System::Void À‡·33::MyForm::textBox4\_TextChanged(System::Object^ sender, System::EventArgs^ e)

{

**return** System::Void();

}

System::Void À‡·33::MyForm::MyForm\_Load(System::Object^ sender, System::EventArgs^ e)

{

**return** System::Void();

}

System::Void À‡·33::MyForm::groupBox2\_Enter(System::Object^ sender, System::EventArgs^ e)

{

**return** System::Void();

}

System::Void À‡·33::MyForm::textBox1\_TextChanged(System::Object^ sender, System::EventArgs^ e)

{

**return** System::Void();

}

System::Void À‡·33::MyForm::domainUpDown1\_SelectedItemChanged(System::Object^ sender, System::EventArgs^ e)

{

**return** System::Void();

}

System::Void À‡·33::MyForm::domainUpDown2\_SelectedItemChanged(System::Object^ sender, System::EventArgs^ e)

{

**return** System::Void();

}

System::Void À‡·33::MyForm::textBox2\_TextChanged(System::Object^ sender, System::EventArgs^ e)

{

**return** System::Void();

}

System::Void À‡·33::MyForm::textBox3\_TextChanged(System::Object^ sender, System::EventArgs^ e)

{

**return** System::Void();

}

MyForm.h

#pragma once

**namespace** Лаб33 {

**using** **namespace** System;

**using** **namespace** System::ComponentModel;

**using** **namespace** System::Collections;

**using** **namespace** System::Windows::Forms;

**using** **namespace** System::Data;

**using** **namespace** System::Drawing;

/// <summary>

/// Сводка для MyForm

/// </summary>

**public** ref **class** MyForm : **public** System::Windows::Forms::Form

{

**public**:

MyForm(**void**)

{

InitializeComponent();

//

//**TODO: добавьте код конструктора**

//

}

**protected**:

/// <summary>

/// Освободить все используемые ресурсы.

/// </summary>

~MyForm()

{

**if** (components)

{

**delete** components;

}

}

**protected**:

**private**: System::Windows::Forms::TextBox^ textBox4;

Convector.cpp

#include "Convector.h"

std::string& Convector\_String\_to\_string(String^ s, std::string& os)

{

**using** **namespace** Runtime::InteropServices;

**const** **char**\* chars =

(**const** **char**\*)(Marshal::StringToHGlobalAnsi(s)).ToPointer();

os = chars;

Marshal::FreeHGlobal(IntPtr((**void**\*)chars));

**return** os;

}

String^ Convector\_string\_to\_String(std::string& os, String^ s)

{

s = gcnew System::String(os.c\_str());

**return** s;

}

System::String^ Convert\_char\_to\_String(**char** ch)

{

**char** chr;

chr = ch;

System::String^ str;

str += **wchar\_t**(ch);

**return** str;

}

#pragma once

#include <iostream>

**using** **namespace** System;

std::string& Convector\_String\_to\_string(String^ s, std::string& os);

System::String^ Convert\_char\_to\_String(**char** ch);

Convector.h

#pragma once

#include <iostream>

**using** **namespace** System;

std::string& Convector\_String\_to\_string(String^ s, std::string& os);

System::String^ Convert\_char\_to\_String(**char** ch);

**Тестирование**

1.Задача коммивояжера