Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Кафедра ІУС

Дисципліна: «Технології об’єктно орієнтованого програмування»

**ПРАКТИЧНА РОБОТА № 6**

**«** **СТВОРЕННЯ ПРОГРАМНИХ ДОДАТКІВ**

**З ВІЗУАЛЬНИМ ІНТЕРФЕЙСОМ»**

|  |  |
| --- | --- |
|  | Прийняла:  Білова Т. Г.  з оцінкою «\_\_»  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_2020р. |

|  |  |
| --- | --- |
|  | Виконав:  Ст. гр. ІТУ-19-2  Куренков Б.М. |

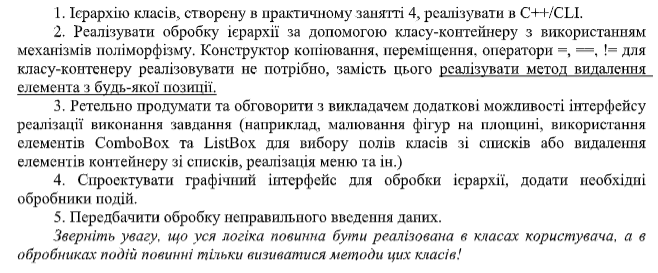
**МЕТА РОБОТИ**

Засвоєння поняття візуального інтерфейсу, набуття навичок реалізації інтерфейсу

користувача для класів та об’єктів.

**ПОСТАНОВКА ЗАДАЧІ**

**Варіант 9**

E:\Media\Media-Transfer\2020-06\opera_e9CbZukKgI.png

**ХІД РОБОТИ**

#pragma once

// структура, которая представляет однозвязный список дуг

struct \_Edge {

int id\_from;

int id\_to;

int weight;

\_Edge\* next;

};

// класс-контейнер для оброботки дуг

class \_Graph {

\_Edge\* head;

public:

// для очистки графа

void Clear\_graph() {

while (head)

this->Pop(head->id\_from, head->id\_to);

}

// конструктор по умолчанию

\_Graph() : head(nullptr) {};

// для добавления дуги в граф

int Add(int id\_from, int id\_to, int weight) {

\_Edge\* tmp = head;

\_Edge\* tmp\_prev = tmp;

if (head)

while (tmp) {

if (tmp->id\_from == id\_from && tmp->id\_to == id\_to) {

if (tmp->weight > weight) {

tmp->weight = weight;

return 0;

}

return 1;

}

tmp\_prev = tmp;

tmp = tmp->next;

}

tmp = new \_Edge;

tmp->id\_from = id\_from;

tmp->id\_to = id\_to;

tmp->weight = weight;

tmp->next = nullptr;

if (!head)

head = tmp;

else

tmp\_prev->next = tmp;

return 2;

}

// для удаления дуги из графа

int Pop(int id\_from, int id\_to) {

if (!head) return 0;

int count = 0;

\_Edge\* tmp = head;

while (tmp->next) {

if (tmp->id\_from == id\_from && tmp->id\_to == id\_to)

break;

tmp = tmp->next;

++count;

}

if (!count) {

if (!tmp->next) {

if (tmp->id\_from != id\_from || tmp->id\_to != id\_to)

return 2;

delete head;

head = nullptr;

return 1;

}

else {

if (tmp->id\_from != id\_from || tmp->id\_to != id\_to)

return 2;

head = tmp->next;

tmp->next = nullptr;

delete tmp;

return 1;

}

}

else {

tmp = head;

\_Edge\* tmp\_prev = tmp;

while (tmp->next) {

if (tmp->id\_from == id\_from && tmp->id\_to == id\_to)

break;

tmp\_prev = tmp;

tmp = tmp->next;

}

if (tmp->id\_from == id\_from && tmp->id\_to == id\_to) {

tmp\_prev->next = tmp->next;

tmp->next = nullptr;

delete tmp;

return 1;

}

return 2;

}

}

// возвращает матрицу весов графа

int\*\* get\_weight\_matrix() {

\_Edge\* tmp = head;

int count\_of\_elems = 0;

while (tmp) {

tmp = tmp->next;

++count\_of\_elems;

}

int\* all\_vertexes = new int[count\_of\_elems \* 2];

tmp = head;

for (int i = 0; i < count\_of\_elems; ++i) {

all\_vertexes[2 \* i] = tmp->id\_from;

all\_vertexes[2 \* i + 1] = tmp->id\_to;

tmp = tmp->next;

}

for (int i = 0, tmp\_swap; i < count\_of\_elems \* 2; ++i) {

for (int j = i; j < count\_of\_elems \* 2; ++j) {

if (all\_vertexes[i] > all\_vertexes[j]) {

tmp\_swap = all\_vertexes[i];

all\_vertexes[i] = all\_vertexes[j];

all\_vertexes[j] = tmp\_swap;

}

}

}

int count\_of\_unique = 0;

for (int i = 0; i < count\_of\_elems \* 2 - 1; ++i)

if (all\_vertexes[i] != all\_vertexes[i + 1])

++count\_of\_unique;

if (count\_of\_unique)

++count\_of\_unique;

if (count\_of\_unique) {

int\*\* weight\_matrix = new int\* [count\_of\_unique];

for (int i = 0; i < count\_of\_unique; ++i)

weight\_matrix[i] = new int[count\_of\_unique];

for (int i = 0; i < count\_of\_unique; ++i)

for (int j = 0; j < count\_of\_unique; ++j)

weight\_matrix[i][j] = 0;

int\* ids = new int[count\_of\_unique];

ids[0] = all\_vertexes[0];

for (int i = 0, j = 0; j < count\_of\_unique - 1; ++i) {

if (all\_vertexes[i] != all\_vertexes[i + 1])

ids[++j] = all\_vertexes[i + 1];

}

tmp = head;

for (int i = 0, id\_from = 0, id\_to = 0; i < count\_of\_elems; ++i, id\_from = 0, id\_to = 0) {

for (int j = 0; j < count\_of\_unique; ++j) {

if (ids[j] == tmp->id\_from)

id\_from = j;

}

for (int j = 0; j < count\_of\_unique; ++j) {

if (ids[j] == tmp->id\_to)

id\_to = j;

}

weight\_matrix[id\_from][id\_to] = tmp->weight;

//weight\_matrix[id\_to][id\_from] = tmp->weight;

tmp = tmp->next;

}

return weight\_matrix;

}

else

return nullptr;

}

// возвращает массив вершин графа

int\* get\_array\_of\_unique() {

int\* array\_of\_unique;

int count\_of\_unique = this->get\_count\_of\_unique();

array\_of\_unique = new int[count\_of\_unique];

\_Edge\* tmp = head;

int count\_of\_elems = 0;

while (tmp) {

tmp = tmp->next;

++count\_of\_elems;

}

int\* all\_vertexes = new int[count\_of\_elems \* 2];

tmp = head;

for (int i = 0; i < count\_of\_elems; ++i) {

all\_vertexes[2 \* i] = tmp->id\_from;

all\_vertexes[2 \* i + 1] = tmp->id\_to;

tmp = tmp->next;

}

for (int i = 0, tmp\_swap; i < count\_of\_elems \* 2; ++i) {

for (int j = i; j < count\_of\_elems \* 2; ++j) {

if (all\_vertexes[i] > all\_vertexes[j]) {

tmp\_swap = all\_vertexes[i];

all\_vertexes[i] = all\_vertexes[j];

all\_vertexes[j] = tmp\_swap;

}

}

}

if (count\_of\_unique) {

int\*\* weight\_matrix = new int\* [count\_of\_unique];

for (int i = 0; i < count\_of\_unique; ++i)

weight\_matrix[i] = new int[count\_of\_unique];

for (int i = 0; i < count\_of\_unique; ++i)

for (int j = 0; j < count\_of\_unique; ++j)

weight\_matrix[i][j] = 0;

array\_of\_unique[0] = all\_vertexes[0];

for (int i = 0, j = 0; j < count\_of\_unique - 1; ++i) {

if (all\_vertexes[i] != all\_vertexes[i + 1])

array\_of\_unique[++j] = all\_vertexes[i + 1];

}

}

return array\_of\_unique;

}

// возвращает кол-во вершин графа

int get\_count\_of\_unique() {

\_Edge\* tmp = head;

int count\_of\_elems = 0;

while (tmp) {

tmp = tmp->next;

++count\_of\_elems;

}

int\* all\_vertexes = new int[count\_of\_elems \* 2];

tmp = head;

for (int i = 0; i < count\_of\_elems; ++i) {

all\_vertexes[2 \* i] = tmp->id\_from;

all\_vertexes[2 \* i + 1] = tmp->id\_to;

tmp = tmp->next;

}

for (int i = 0, tmp\_swap; i < count\_of\_elems \* 2; ++i) {

for (int j = i; j < count\_of\_elems \* 2; ++j) {

if (all\_vertexes[i] > all\_vertexes[j]) {

tmp\_swap = all\_vertexes[i];

all\_vertexes[i] = all\_vertexes[j];

all\_vertexes[j] = tmp\_swap;

}

}

}

int count\_of\_unique = 0;

for (int i = 0; i < count\_of\_elems \* 2 - 1; ++i)

if (all\_vertexes[i] != all\_vertexes[i + 1])

++count\_of\_unique;

if (count\_of\_unique)

++count\_of\_unique;

return count\_of\_unique;

}

// возвращает кол-во компонент связности графа

int get\_connectivity\_component\_count() {

int count = 0;

int size = this->get\_count\_of\_unique();

int\*\* matrix = this->get\_weight\_matrix();

bool\* used = new bool[size];

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

used[i] = 0;

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

if (used[i] == 0) {

this->dfs(i, used, matrix, size);

++count;

}

}

return count;

}

// возвращает кол-во рёбер

int get\_count\_of\_edges() {

int count = 0;

\_Edge\* tmp = this->head;

while (tmp) {

++count;

tmp = tmp->next;

}

return count;

}

};

-----------------------------------------------------------------------------------------

graph = new \_Graph();

// запрещает пользователю вводить что-либо, кроме цифр

private: System::Void only\_num\_KeyPress(System::Object^ sender, System::Windows::Forms::KeyPressEventArgs^ e) {

if (!Char::IsDigit(e->KeyChar) && e->KeyChar != 0x08)

e->Handled = true;

}

// запрещает пользователю вставлять информацию через комбинацию клавиш Ctrl+V

private: System::Void deny\_paste\_KeyDown(System::Object^ sender, System::Windows::Forms::KeyEventArgs^ e) {

((TextBox^)sender)->ContextMenu = nullptr;

((TextBox^)sender)->ContextMenuStrip = nullptr;

}

//вспомогательная функция для событий add\_edge\_Click и delete\_edge\_Click. обновляет нек-рые информационные поля

private: void info\_update() {

String^ info, ^ weight\_matrix;

info += "Кол-во вершин: " + graph->get\_count\_of\_unique().ToString() + "\r\n";

info += "Кол-во дуг: " + graph->get\_count\_of\_edges().ToString() + "\r\n";

int\*\* weights = graph->get\_weight\_matrix();

int count\_of\_unique = graph->get\_count\_of\_unique();

int\* array\_of\_unique = graph->get\_array\_of\_unique();

weight\_matrix += " ";

for (int i = 0; i < count\_of\_unique; ++i) {

weight\_matrix += array\_of\_unique[i].ToString() + " ";

}

weight\_matrix += "\r\n";

for (int i = 0; i < count\_of\_unique; ++i, weight\_matrix += "\r\n") {

weight\_matrix += array\_of\_unique[i].ToString() + " ";

for (int j = 0; j < count\_of\_unique; ++j) {

weight\_matrix += weights[i][j].ToString() + " ";

}

}

info += "Кол-во компонент связности: " + graph->get\_connectivity\_component\_count().ToString();

info\_textBox->Text = info;

}

// для добавления дуги в граф

private: System::Void add\_edge\_Click(System::Object^ sender, System::EventArgs^ e) {

String^ cpy, ^log;

if (!source\_textBox->Text->Length || !target\_textBox->Text->Length || !weight\_textBox->Text->Length) {

info\_textBox->Text += "Отказ! (пустые поля)\r\n";

}

else if (source\_textBox->Text == target\_textBox->Text) {

info\_textBox->Text += "Отказ! (петля)\r\n";

}

else if (weight\_textBox->Text == "0") {

cpy = log\_textBox->Text;

log\_textBox->Text = "";

log\_textBox->Text += "Отказ! (нулевой вес)\r\n";

log\_textBox->Text += cpy;

}

else {

String^ id\_from = source\_textBox->Text;

String^ id\_to = target\_textBox->Text;

String^ \_weight = weight\_textBox->Text;

int source, target, weight;

source = source.Parse(id\_from);

target = target.Parse(id\_to);

weight = weight.Parse(\_weight);

int res = graph->Add(source, target, weight);

log = "Дуга " + id\_from + " - > " + id\_to + " с весом " + \_weight;

switch (res) {

case 0:

log += " перезаписана!\r\n"; break;

case 1:

log += " не перезаписана!\r\n"; break;

case 2:

log += " добавлена!\r\n"; break;

}

info\_update();

}

}

// для удаления дуги из графа

private: System::Void delete\_edge\_Click(System::Object^ sender, System::EventArgs^ e) {

String^ cpy, ^ log;

if (!source\_textBox->Text->Length || !target\_textBox->Text->Length || !weight\_textBox->Text->Length) {

info\_textBox->Text += "Отказ! (пустые поля)\r\n";

}

else if (source\_textBox->Text == target\_textBox->Text) {

info\_textBox->Text += "Отказ! (петля)\r\n";

}

else {

String^ id\_from = source\_textBox->Text;

String^ id\_to = target\_textBox->Text;

int source, target;

source = source.Parse(id\_from);

target = target.Parse(id\_to);

int res = graph->Pop(source, target);

log = "Дуга " + id\_from + " - > " + id\_to;

switch (res) {

case 0:

log += " не удалёна, ведь граф уже пуст!\r\n"; break;

case 1:

log += " удалёна!\r\n"; break;

case 2:

log += " не найдена!\r\n"; break;

}

info\_update();

}

}

// для очистки графа от дуг

private: System::Void clear\_graph\_button\_Click(System::Object^ sender, System::EventArgs^ e) {

graph->Clear\_graph();

info\_textBox->Text = "";

weight\_matrix\_textBox->Text = "";

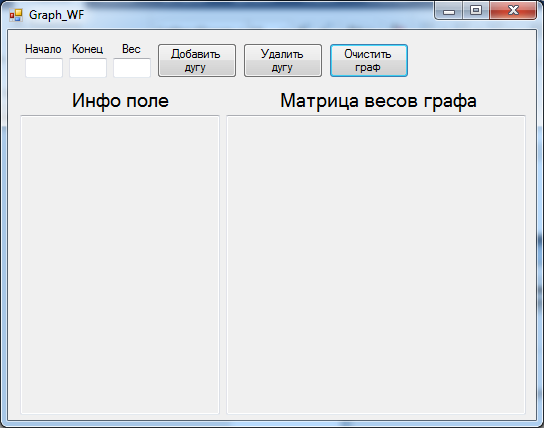
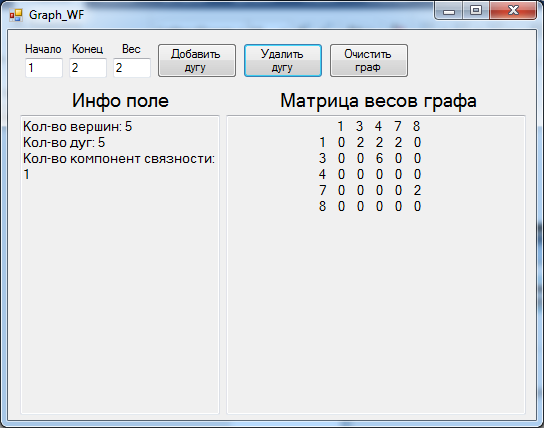
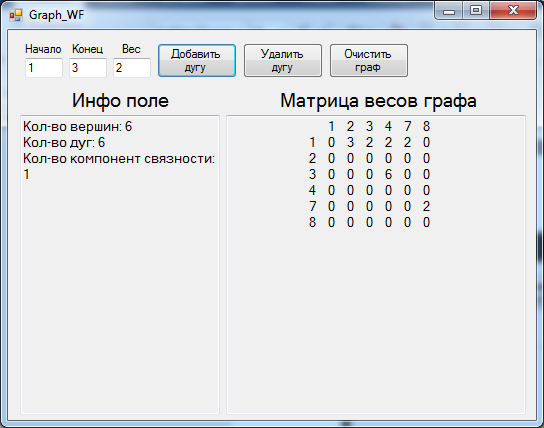
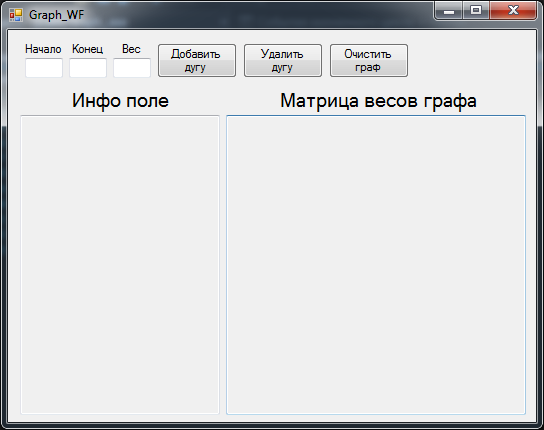
source\_textBox->Text = "";

target\_textBox->Text = "";

weight\_textBox->Text = "";

}

**ВИКОНАННЯ ЕКСПЕРИМЕНТІВ ПО НАЛАГОДЖЕННЮ ПРОГРАМИ**

****

**Висновки.**

Протягом лабораторної роботи було розроблено программу, яка здатна напомнювати граф дугами, обробляти їх та виводити на екран матрицю ваг.