Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Кафедра ІУС

Дисципліна: «Технології об’єктно орієнтованого програмування»

**ПРАКТИЧНА РОБОТА № 5**

**«** **ШАБЛОНИ КЛАСІВ ТА ОБРОБКА ВИКЛЮЧЕНЬ»**

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

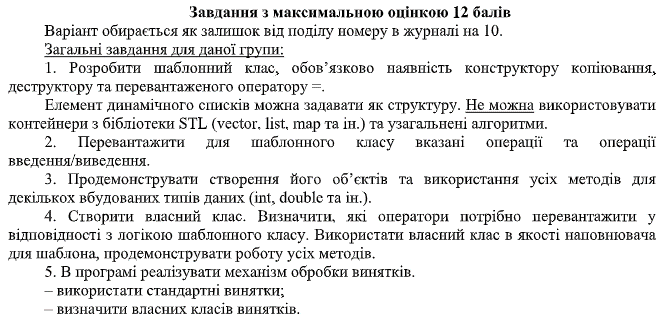
|  |  |
| --- | --- |
|  | Виконав:  Ст. гр. ІТУ-19-2  Куренков Б.М. |

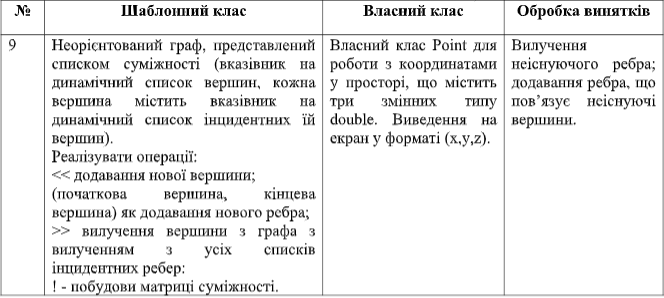
**МЕТА РОБОТИ**

Засвоєння поняття шаблонних класів та винятків; набуття практичних навичок їх програмування.

**ПОСТАНОВКА ЗАДАЧІ**

**Варіант 9**





**ХІД РОБОТИ**

#pragma once

#include <iostream>

#include <iomanip>

class Point {

float x, y, z;

public:

Point() : x(0), y(0), z(0) {}

Point(float x, float y, float z) : x(x), y(y), z(z) {}

void setCord(float y, float x, float z) {

this->x = x; this->y = y; this->z = z;

}

float\* getCord() const {

float\* tmp = new float[3];

tmp[0] = x; tmp[1] = y; tmp[2] = z;

return tmp;

}

bool operator==(Point& p) {

if (&p == this) return 1;

if (p.x == x && p.y == y && p.z == z) return 1;

return 0;

}

bool operator!=(Point& p) {

return !(p == \*this);

}

bool operator>(Point& p) {

if ((x + y + z) > (p.x + p.y + p.z))

return 1;

return 0;

}

friend std::istream& operator>>(std::istream& in, Point& point);

};

std::ostream& operator<<(std::ostream& out, Point& point) {

float\* tmp = point.getCord();

out << "(" << tmp[0] << ", " << tmp[1] << ", " << tmp[2] << ")";

return out;

}

class Graph {

public:

struct Elem {

private:

Elem() :it(), has(NULL), next(NULL) {}

public:

Point it;

Elem\* has;

Elem\* next;

Elem(Point it) :it(it), has(NULL), next(NULL) {}

~Elem() {

Elem\* tmp = this;

Elem\* del;

while (tmp->has) {

del = this;

while (del->has->has) {

del = del->has;

}

delete del->has;

del->has = NULL;

}

}

};

public:

Graph() : data(NULL) {}

Graph(Graph::Elem\* data) {

Elem\* tmp = new Elem(\*(data));

this->data = tmp;

}

Graph(Graph& graph) {

Elem\* tmp = new Elem(\*(graph.data));

this->data = tmp;

}

Graph& operator=(Graph& graph) {

if (&graph == this) return \*this;

Elem\* tmp = new Elem(\*(graph.data));

this->data = tmp;

return \*this;

}

~Graph() {

Elem\* tmp = data;

Elem\* del;

while (tmp->next) {

del = data;

while (del->next->next) {

del = del->next;

}

del->next->~Elem();

delete del->next;

del->next = NULL;

}

data->~Elem();

delete data;

data = NULL;

}

void operator<<(Graph::Elem\* elems[2]) {

if (!data)

data = new Elem({ 0,0,0 });

Elem\* tmp;

Point point\_tmp;

if (data->it == point\_tmp) {

data->it = elems[0]->it;

tmp = data;

while (tmp->has)

tmp = tmp->has;

tmp->has = new Elem(elems[1]->it);

data->next = new Elem(elems[1]->it);

}

else {

tmp = data;

while (tmp->it != elems[0]->it && tmp->next)

tmp = tmp->next;

if (tmp->it == elems[0]->it) {

while (tmp->has)

tmp = tmp->has;

tmp->has = new Elem(elems[1]->it);

}

else {

tmp->next = new Elem(elems[0]->it);

tmp->next->has = new Elem(elems[1]->it);

}

tmp = data;

while (tmp->next && tmp->it != elems[1]->it)

tmp = tmp->next;

if (tmp->it != elems[1]->it) {

tmp->next = new Elem(elems[1]->it);

}

}

}

void operator>>(Graph::Elem& elem) {

Elem\* tmp, \* tmp\_prev = NULL, \* tmp\_next, \* tmp\_has, \* tmp\_has\_prev, \* tmp\_has\_next;

if (!data) { std::cout << "empty!\n\n"; }

else {

tmp = data;

while (tmp) {

if (data->it == elem.it) {

data->~Elem();

tmp = data->next;

delete data;

this->data = tmp;

continue;

}

else if (tmp->it == elem.it) {

tmp->~Elem();

tmp\_next = tmp->next;

delete tmp;

tmp\_prev->next = tmp\_next;

tmp = tmp\_next;

continue;

}

tmp\_has\_prev = tmp;

tmp\_has = tmp->has;

while (tmp\_has) {

if (tmp\_has->it == elem.it) {

tmp\_has\_prev->has = tmp\_has->has;

tmp\_has->has = NULL;

delete tmp\_has;

tmp\_has = NULL;

tmp\_has = tmp\_has\_prev->has;

continue;

}

tmp\_has\_prev = tmp\_has;

tmp\_has = tmp\_has->has;

}

tmp\_prev = tmp;

tmp = tmp->next;

}

}

}

int\*\* operator!() {

if (!data) { std::cout << "emtpty!\n\n"; return 0; }

else {

Elem\* tmp, \* tmp\_prev, \* tmp\_next, \* tmp\_has, \* tmp\_has\_prev, \* tmp\_has\_next;

tmp = data;

while (tmp) {

if (!tmp->has) {

int flag = 1;

tmp\_has = data;

tmp\_next = data;

while (tmp\_next) {

tmp\_has\_next = tmp\_next->has;

while (tmp\_has\_next) {

if (tmp\_has\_next->it == tmp->it) {

flag = 0; break;

}

tmp\_has\_next = tmp\_has\_next->has;

}

tmp\_next = tmp\_next->next;

}

try {

if (flag)

throw 1;

}

catch (int f) {

if (f) {

\*this >> \*(new Elem(tmp->it));

tmp = data;

}

}

}

tmp = tmp->next;

}

tmp = data, tmp\_has = NULL;

int count = 0;

while (tmp) {

tmp = tmp->next;

++count;

}

int\*\* adjacency\_matrix = new int\* [count];

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

adjacency\_matrix[i] = new int[count];

}

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

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

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

}

}

Point\* ids = new Point[count];

tmp = data;

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

ids[i] = tmp->it;

tmp = tmp->next;

}

tmp = data;

tmp\_has = tmp->has;

for (int i = 0; tmp; ++i) {

tmp\_has = tmp->has;

for (int k = 0; tmp\_has;) {

while (1) {

if (ids[k] == tmp\_has->it)

break;

++k;

}

adjacency\_matrix[i][k] += 1;

adjacency\_matrix[k][i] += 1;

k = 0;

tmp\_has = tmp\_has->has;

}

tmp = tmp->next;

}

Point p\_tmp({ 0,0,0 });

for (int i = 0, tmp; i < count; ++i) {

for (int j = i; j < count; ++j) {

if (ids[i] > ids[j]) {

for (int a = 0; a < count; ++a) {

tmp = adjacency\_matrix[a][i];

adjacency\_matrix[a][i] = adjacency\_matrix[a][j];

adjacency\_matrix[a][j] = tmp;

}

for (int a = 0; a < count; ++a) {

tmp = adjacency\_matrix[i][a];

adjacency\_matrix[i][a] = adjacency\_matrix[j][a];

adjacency\_matrix[j][a] = tmp;

}

p\_tmp = ids[i];

ids[i] = ids[j];

ids[j] = p\_tmp;

}

}

}

delete[] ids;

return adjacency\_matrix;

}

}

Elem\* get\_data() { return data; }

private:

Elem\* data;

};

std::ostream& operator<<(std::ostream& out, Graph& graph) {

Graph::Elem\* tmp = graph.get\_data(),\* tmp\_has;

while (tmp) {

tmp\_has = tmp->has;

while (tmp\_has) {

out << "x" << tmp->it << " -> x" << tmp\_has->it << ", ";

tmp\_has = tmp\_has->has;

}

out << '\n';

tmp = tmp->next;

}

out << '\n';

out << '\n';

tmp = graph.get\_data();

int count = 0;

while (tmp) {

tmp = tmp->next;

++count;

}

int\*\* adjacency\_matrix = !graph;

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

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

out << std::setw(4) << adjacency\_matrix[i][j];

}

out << '\n';

out << '\n';

}

out << '\n';

out << '\n';

return out;

}

std::istream& operator>>(std::istream& in, Graph& graph) {

Point ar, br;

in >> ar >> br;

Graph::Elem\* a = new Graph::Elem(ar);

Graph::Elem\* b = new Graph::Elem(br);

Graph::Elem\* ab[2] = { a, b };

graph << ab;

return in;

}

std::istream& operator>>(std::istream& in, Point& point) {

in >> point.x >> point.y >> point.z;

return in;

}

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

#include "Undirected graph.h"

int main() {

Point p1(1, 2, 3), p2(2, 3, 4), p3(3, 4, 5), p4(4, 5, 6), p5(5, 6, 7), p6(6, 7, 8);

Graph::Elem\* t1 = new Graph::Elem(p1), \* t2 = new Graph::Elem(p2), \* t3 = new Graph::Elem(p3),

\* t4 = new Graph::Elem(p4), \* t5 = new Graph::Elem(p5), \* t6 = new Graph::Elem(p6);

Graph::Elem\* t12[2] = { t1, t2 };

Graph::Elem\* t34[2] = { t3, t4 };

Graph::Elem\* t43[2] = { t4, t3 };

Graph::Elem\* t13[2] = { t1, t3 };

Graph::Elem\* t45[2] = { t4, t5 };

Graph::Elem\* t24[2] = { t2, t4 };

Graph::Elem\* t42[2] = { t4, t2 };

Graph::Elem\* t46[2] = { t4, t6 };

Graph::Elem\* t11[2] = { t1, t1 };

Graph\* g4 = new Graph;

\*g4 << t12;

\*g4 << t43;

\*g4 << t34;

\*g4 << t45;

\*g4 << t24;

\*g4 << t42;

\*g4 << t46;

\*g4 << t11;

std::cin >> \*g4;

std::cin >> \*g4;

std::cout << \*g4;

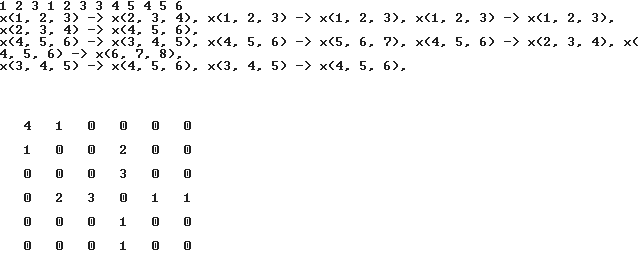
std::cout << '\n';

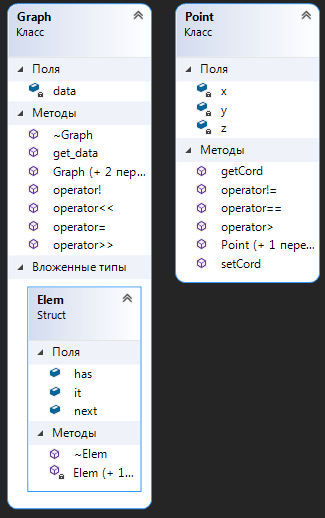
std::cout << '\n';

return 0;

}

**ВИКОНАННЯ ЕКСПЕРИМЕНТІВ ПО НАЛАГОДЖЕННЮ ПРОГРАМИ**

****



Діаграма классів

**Висновки.**

Протягом лабораторної роботи було розроблено программу, яка здатна напомнювати граф єлементами у трьохвимірній координатній системі, обробляти їх та виводити на єкран множину сміжності та матрицю сміжності.