Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Кафедра ІУС

Дисципліна: «Технології об’єктно орієнтованого програмування»

**ПРАКТИЧНА РОБОТА № 3**

**«ПЕРЕВАНТАЖЕННЯ ОПЕРАЦІЙ»**

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

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

**МЕТА РОБОТИ**

Засвоєння поняття перевантаження операцій, засвоєння навичок перевантаження унарних та бінарних операцій методами класу та за допомогою дружніх функцій.

**ПОСТАНОВКА ЗАДАЧІ**

**Завдання з максимальною оцінкою 10 балів**

Варіант 9

Загальні завдання для даної групи:

1. Реалізувати сутності з динамічно розміщеним списком елементів (в якості елементів використовувати ВКЛАДЕНІ КЛАСИ, див. приклад 3 в методичних вказівках до ПЗ).

2. Включити в клас конструктор за замовченням, конструктор, який використовує в якості одного з параметрів одновимірний або двовимірний масив чисел (за потребою з контролем на існування об’єкту відповідного типу).

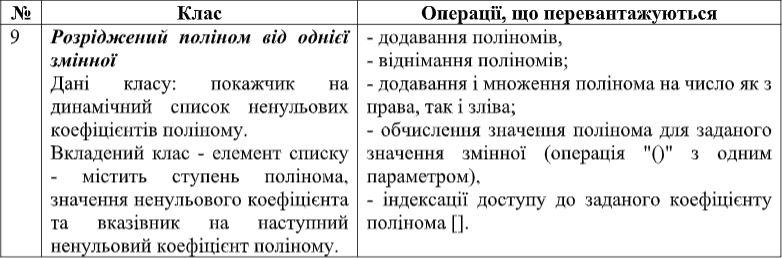
3. Реалізувати деструктор, конструктор копіювання, конструктор переміщення, та перевантажити оператор присвоювання(=).

4. Реалізувати необхідні методи додавання та видалення елементів.

5. Перевантажити для класу оператор виведення (<<), оператор порівняння (==) та оператор нерівності (!=).

6. Перевантажити вказані для варіанта операції.

7. В головній програмі продемонструвати роботу всіх методів та функцій.



**ХІД РОБОТИ**

#pragma once

#include <iostream> // раз уж все комментим)

#include <utility> // для std::move()

class Polynomial;

class Polynomial\_info { // информация о полиноме

int power; // его степень

float coef; // его коефициент

Polynomial\_info\* next; // указатель на следующих "х"

Polynomial\_info() = default; // конструктор по умолчани

public:

~Polynomial\_info(); // деструкторю

Polynomial\_info(int power, float coef); // конструктор с параметрами

void show();

friend std::ostream& operator<<(std::ostream& out, const Polynomial& pol); // для вывода информации о полиноме

friend class Polynomial;

friend Polynomial& operator\*(float mult\_coef, const Polynomial& pol); // для умножения полинома на число слева

friend Polynomial& operator\*(const Polynomial& pol, float mult\_coef); // для умножения полинома на число справа

};

class Polynomial { // полином

static int count\_of\_pols; // количество существующих полиномов

static int current\_count\_of\_pols;

int p\_num; // номер полинома

Polynomial\_info\* data; // ссылка на информацию о полиноме

int count\_of\_variables; // кол-во "х-ов"

public:

~Polynomial(); // деструктор

Polynomial(); // конструктор по умолчанию

Polynomial(const int size, Polynomial\_info\*\* datas); // конструктор с параметрами "

template <char ch = '+'>

int push(const int size, Polynomial\_info\* datas = NULL);

template <char ch = '+'>

int push(const int size, Polynomial\_info\*\* datas = NULL);

Polynomial(Polynomial& pol); // конструктор копирования

Polynomial& operator=(Polynomial& pol); // оператор копирования

Polynomial(Polynomial&& pol); // конструктор перемещения

Polynomial& operator=(Polynomial&& pol); // оператор перемещения

bool operator==(const Polynomial& pol); // для сравнения полиномов на идентичность

bool operator!=(const Polynomial& pol); // для сравнения полиномов на разность

Polynomial& operator+(Polynomial& pol); // для сложения полиномов

Polynomial& operator-(Polynomial& pol); // для вычитания полиномов

float operator()(float x); // для возврата значения полинома при заданном "х"

float operator[](int id); // для возврата коефициента "х" при указанном id

friend Polynomial& operator+(Polynomial\_info elem, const Polynomial& pol); // для сложения полинома с числом слева

friend Polynomial& operator+(const Polynomial& pol, Polynomial\_info elem); // для сложения полинома с числом справа

friend Polynomial& operator\*(float mult\_coef, const Polynomial& pol); // для умножения полинома на число слева

friend Polynomial& operator\*(const Polynomial& pol, float mult\_coef); // для умножения полинома на число справа

friend std::ostream& operator<<(std::ostream& out, const Polynomial& pol); // для вывода информации о полиноме

friend class Polynomial\_info; //

};

int Polynomial::count\_of\_pols = 0;

int Polynomial::current\_count\_of\_pols = 0;

Polynomial\_info::~Polynomial\_info() {

next = NULL;

}

Polynomial\_info::Polynomial\_info(int power, float coef) {

power >= 0 ? this->power = power : this->power = -1 \* power;

this->coef = coef;

this->next = NULL;

}

void Polynomial\_info::show() {

std::cout << "Power = " << power << ", coef = " << coef << ", next = " << next << "\n\n";

}

//---------------------------------------------------------------------------------------

Polynomial::~Polynomial() { // деструктор

--current\_count\_of\_pols;

data = NULL;

}

Polynomial::Polynomial() { // конструктор по умолчанию

p\_num = ++count\_of\_pols;

++current\_count\_of\_pols;

data = NULL;

count\_of\_variables = 0;

}

Polynomial::Polynomial(const int size, Polynomial\_info\*\* datas) {

p\_num = ++count\_of\_pols;

++current\_count\_of\_pols;

if (datas)

this->push(size, datas);

else this->data = NULL;

}

template <char ch>

int Polynomial::push(const int size, Polynomial\_info\* datas) { // пуш элементов

int i = 0;

if (datas == NULL) return 0;

if (ch != '-') {

if (data == NULL) {

data = new Polynomial\_info(datas->power, datas->coef);

++i;

datas = datas->next;

}

Polynomial\_info\* tmp;

for (; i < size && datas != NULL; ++i, datas = datas->next) {

tmp = this->data;

while (tmp->next != NULL && datas->power != tmp->power) {

tmp = tmp->next;

}

if (datas->power == tmp->power) {

tmp->coef += datas->coef;

}

else {

tmp->next = new Polynomial\_info;

tmp->next->power = datas->power;

tmp->next->coef = datas->coef;

}

}

}

else {

if (data == NULL) {

data = new Polynomial\_info(datas->power, -1. \* (datas->coef));

++i;

datas = datas->next;

}

Polynomial\_info\* tmp;

for (; i < size && datas != NULL; ++i, datas = datas->next) {

tmp = this->data;

while (tmp->next != NULL && datas->power != tmp->power) {

tmp = tmp->next;

}

if (datas->power == tmp->power) {

tmp->coef -= datas->coef;

}

else {

tmp->next = new Polynomial\_info;

tmp->next->power = datas->power;

tmp->next->coef = -1. \* (datas->coef);

}

}

}

Polynomial\_info\* tmp = this->data;

int iter = 0;

while (tmp && tmp->coef == 0) {

this->data = this->data->next;

tmp = this->data;

}

if (tmp) {

++iter;

while (tmp->next != NULL) {

tmp = tmp->next;

++iter;

}

}

count\_of\_variables = iter;

return i;

}

template <char ch>

int Polynomial::push(const int size, Polynomial\_info\*\* datas) { // пуш элементов

int i = 0;

if (datas == NULL) return 0;

if ((\*datas) == NULL) return 0;

if (ch != '-') {

if (data == NULL) {

data = new Polynomial\_info(datas[i]->power, datas[i]->coef);

++i;

}

Polynomial\_info\* tmp;

for (; i < size; ++i) {

tmp = this->data;

while (tmp->next != NULL && datas[i]->power != tmp->power) {

tmp = tmp->next;

}

if (datas[i]->power == tmp->power) {

tmp->coef += datas[i]->coef;

}

else {

tmp->next = new Polynomial\_info;

tmp->next->power = datas[i]->power;

tmp->next->coef = datas[i]->coef;

}

}

}

else {

if (data == NULL) {

data = new Polynomial\_info(datas[i]->power, -1. \* (datas[i]->coef));

++i;

}

Polynomial\_info\* tmp;

for (; i < size; ++i) {

tmp = this->data;

while (tmp->next != NULL && datas[i]->power != tmp->power) {

tmp = tmp->next;

}

if (datas[i]->power == tmp->power) {

tmp->coef -= datas[i]->coef;

}

else {

tmp->next = new Polynomial\_info;

tmp->next->power = datas[i]->power;

tmp->next->coef = -1. \* (datas[i]->coef);

}

}

}

Polynomial\_info\* tmp = this->data;

int iter = 0;

while (tmp && tmp->coef == 0) {

this->data = this->data->next;

tmp = this->data;

}

if (tmp) {

++iter;

while (tmp->next != NULL) {

tmp = tmp->next;

++iter;

}

}

count\_of\_variables = iter;

return i;

}

Polynomial::Polynomial(Polynomial& pol) { // конструктор копирования

this->count\_of\_variables = pol.count\_of\_variables;

p\_num = ++count\_of\_pols;

++current\_count\_of\_pols;

if (pol.data) this->push(pol.count\_of\_variables, pol.data);

}

Polynomial& Polynomial::operator=(Polynomial& pol) { // оператор копирования

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

this->p\_num = count\_of\_pols;

Polynomial\* tmp = new Polynomial;

--tmp->current\_count\_of\_pols;

--tmp->count\_of\_pols;

if (pol.data) tmp->push(pol.count\_of\_variables, pol.data);

this->count\_of\_variables = tmp->count\_of\_variables;

this->data = tmp->data;

return \*this;

}

Polynomial::Polynomial(Polynomial&& pol) { // конструктор перемещения

if (this != &pol) {

this->data = pol.data;

this->p\_num = ++count\_of\_pols;

count\_of\_variables = pol.count\_of\_variables;

pol.count\_of\_variables = 0;

pol.data = NULL;

}

}

Polynomial& Polynomial::operator=(Polynomial&& pol) { // оператор перемещения

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

this->data = pol.data;

count\_of\_variables = pol.count\_of\_variables;

pol.count\_of\_variables = 0;

pol.data = NULL;

return \*this;

}

bool Polynomial::operator==(const Polynomial& pol) { // для сравнения полиномов на идентичность

if (this == &pol) return 1;

if (this->count\_of\_variables == pol.count\_of\_variables) return 1; // tmp->data == NULL || pol.data == NULL included

Polynomial\_info\* tmp;

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

tmp = this->data;

while (tmp->next != NULL && tmp->power != pol.data->power)

tmp = tmp->next;

if (!(tmp->power == pol.data->power && tmp->coef == pol.data->coef)) return 0;

}

return 1;

}

bool Polynomial::operator!=(const Polynomial& pol) { // для сравнения полиномов на разность

if (this == &pol) return 0;

if (this->count\_of\_variables != pol.count\_of\_variables) return 1; // tmp->data == NULL || pol.data == NULL included

Polynomial\_info\* tmp;

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

tmp = this->data;

while (tmp->next != NULL && tmp->power != pol.data->power)

tmp = tmp->next;

if (tmp->power == pol.data->power && tmp->coef == pol.data->coef) return 0;

}

return 1;

}

Polynomial& Polynomial::operator+(Polynomial& pol) { // для сложения полиномов

Polynomial\* tmp = new Polynomial;

--tmp->current\_count\_of\_pols;

--tmp->count\_of\_pols;

tmp->push(this->count\_of\_variables, this->data);

tmp->push(pol.count\_of\_variables, pol.data);

return \*tmp;

}

Polynomial& Polynomial::operator-(Polynomial& pol) { // для вычитания полиномов

Polynomial\* tmp = new Polynomial;

--tmp->current\_count\_of\_pols;

--tmp->count\_of\_pols;

tmp->push<'+'>(this->count\_of\_variables, this->data);

tmp->push<'-'>(pol.count\_of\_variables, pol.data);

return \*tmp;

}

float Polynomial::operator()(float x) { // для возврата значения полинома при заданном "х"

if (!data) exit(1);

float sum = 0;

Polynomial\_info\* tmp = this->data;

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

sum += tmp->coef \*pow(x, tmp->power);

tmp = tmp->next;

}

return sum;

}

float Polynomial::operator[](int id) { // для возврата коефициента "х" при указанном id

if (!data) exit(2);

id < 0 ? id \*= -1 : id = id;

if (!(id < count\_of\_variables)) exit(3);

float coef;

Polynomial\_info\* tmp = this->data;

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

tmp = tmp->next;

coef = tmp->coef;

return coef;

}

Polynomial& operator+(Polynomial\_info elem, const Polynomial& pol) { // для сложения полинома с числом слева

Polynomial\* tmp = new Polynomial;

--tmp->current\_count\_of\_pols;

--tmp->count\_of\_pols;

if (pol.data)

tmp->push(pol.count\_of\_variables, pol.data);

tmp->push(1, &elem);

return \*tmp;

}

Polynomial& operator+(const Polynomial& pol, Polynomial\_info elem) { // для сложения полинома с числом справа

Polynomial\* tmp = new Polynomial;

--tmp->current\_count\_of\_pols;

--tmp->count\_of\_pols;

if (pol.data)

tmp->push(pol.count\_of\_variables, pol.data);

tmp->push(1, &elem);

return \*tmp;

}

Polynomial& operator\*(float mult\_coef, const Polynomial& pol) { // для умножения полинома на число слева

Polynomial\* tmp = new Polynomial;

--tmp->current\_count\_of\_pols;

--tmp->count\_of\_pols;

if (pol.data) {

tmp->push(pol.count\_of\_variables, pol.data);

Polynomial\_info\* tmp\_for\_mult = tmp->data;

for (int i = 0; i < tmp->count\_of\_variables; ++i) {

tmp\_for\_mult->coef \*= mult\_coef;

tmp\_for\_mult = tmp\_for\_mult->next;

}

}

return \*tmp;

}

Polynomial& operator\*(const Polynomial& pol, float mult\_coef) { // для умножения полинома на число справа

Polynomial\* tmp = new Polynomial;

--tmp->current\_count\_of\_pols;

--tmp->count\_of\_pols;

if (pol.data) {

tmp->push(pol.count\_of\_variables, pol.data);

Polynomial\_info\* tmp\_for\_mult = tmp->data;

for (int i = 0; i < tmp->count\_of\_variables; ++i) {

tmp\_for\_mult->coef \*= mult\_coef;

tmp\_for\_mult = tmp\_for\_mult->next;

}

}

return \*tmp;

}

std::ostream& operator<<(std::ostream& out, const Polynomial& pol) { // для вывода информации о полиноме

out << "Polynomial #" << pol.p\_num << " contains " << pol.count\_of\_variables << " variables : ";

Polynomial\_info\* tmp = pol.data;

if (tmp == NULL) {

out << "0";

out << "\nCurrent count of polynominals: " << pol.current\_count\_of\_pols << "\n\n";

return out;

}

else {

if (tmp->coef < 0) { // меньше нуля

switch (tmp->power) {

case 0: // -...\*1

out << tmp->coef;

tmp = tmp->next;

break;

case 1: // -...\*х

if (tmp->coef == 1.) { // -1\*х

out << "-x";

tmp = tmp->next;

break;

}

else { // -1.1\*х

out << tmp->coef << "x";

tmp = tmp->next;

break;

}

default: // -...\*х^12

if (tmp->coef == 1.) { // -1\*х^12

out << "-x^" << tmp->power;

tmp = tmp->next;

break;

}

else { // -1.1\*х^12

out << tmp->coef << "x^" << tmp->power;

tmp = tmp->next;

break;

}

}

}

else { // больше нуля

switch (tmp->power) {

case 0: // ...\*1

out << tmp->coef;

tmp = tmp->next;

break;

case 1: // ...\*x

if (tmp->coef == 1.) { // 1\*x

out << "x";

tmp = tmp->next;

break;

}

else { // 1.1\*x

out << tmp->coef << "x";

tmp = tmp->next;

break;

}

default: // ...\*x^12

if (tmp->coef == 1.) { // 1\*x^12

out << "x^" << tmp->power;

tmp = tmp->next;

break;

}

else { // 1.2\*x^12

out << tmp->coef << "x^" << tmp->power;

tmp = tmp->next;

break;

}

}

}

while (tmp != NULL) {

if (tmp->coef < 0) { // меньше нуля

switch (tmp->power) {

case 0: // -...\*1

out << " - " << abs(tmp->coef);

tmp = tmp->next;

break;

case 1: // -...\*х

if (tmp->coef == -1.) { // -1\*х

out << " - x";

tmp = tmp->next;

break;

}

else { // -1.1\*х

out << " - " << abs(tmp->coef) << "x";

tmp = tmp->next;

break;

}

default: // -...\*х^12

if (tmp->coef == -1.) { // -1\*х^12

out << " - x^" << tmp->power;

tmp = tmp->next;

break;

}

else { // -1.1\*х^12

out << " - " << abs(tmp->coef) << "x^" << tmp->power;

tmp = tmp->next;

break;

}

}

}

else { // больше нуля

switch (tmp->power) {

case 0: // ...\*1

out << " + " << tmp->coef;

tmp = tmp->next;

break;

case 1: // ...\*x

if (tmp->coef == 1.) { // 1\*x

out << " + x";

tmp = tmp->next;

break;

}

else { // 1.1\*x

out << " + " << tmp->coef << "x";

tmp = tmp->next;

break;

}

default: // ...\*x^12

if (tmp->coef == 1.) { // 1\*x^12

out << " + " << "x^" << tmp->power;

tmp = tmp->next;

break;

}

else { // 1.1\*x^12

out << " + " << tmp->coef << "x^" << tmp->power;

tmp = tmp->next;

break;

}

}

}

}

}

out << "\nCurrent count of polynominals: " << pol.current\_count\_of\_pols << "\n\n";

return out;

}

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

#include "Polynomial.h"

int main() {

setlocale(LC\_ALL, "Russian");

Polynomial\_info\* data0 = new Polynomial\_info(7, 1.1);

Polynomial\_info\* data1 = new Polynomial\_info(7, -2.2);

Polynomial\_info\* data2 = new Polynomial\_info(0, -3.3);

Polynomial\_info\* data3 = new Polynomial\_info(1, 4.4);

Polynomial\_info\* data4 = new Polynomial\_info(7, 5.);

Polynomial\_info\* data5 = new Polynomial\_info(1, -6.);

Polynomial\_info\* data6 = new Polynomial\_info(0, 7.);

Polynomial\_info\* data7 = new Polynomial\_info(5, -8.);

Polynomial\_info\* data8 = new Polynomial\_info(1, -9.);

Polynomial\_info\* datas0[4] = { data2 , data1, data0, data3 };

Polynomial\_info\* datas1[4] = { data0 , data1, data2, data3 };

Polynomial\_info\* datas2[5] = { data4 , data5, data6, data7, data8 };

Polynomial\* pol1 = new Polynomial;

std::cout << pol1->push(4, datas1) << std::endl;

std::cout << pol1->push(5, datas2) << std::endl;

std::cout << \*pol1;

Polynomial\* pol2 = new Polynomial;

pol2->push(3, datas2);

std::cout << \*pol2;

std::cout << "\n\n";

Polynomial pol3 = \*pol1 + \*pol2;

std::cout << pol3;

pol3 = pol3 - \*pol2;

Polynomial\_info\* data04 = new Polynomial\_info(15, 4.4);

pol3 = pol3 + \*data04;

std::cout << pol3;

std::cout << "\n\nDoes pol1 and pol3 are equal?";

\*pol1 == \*pol2 ? std::cout << "\nPolynomials are equal!!!\n\n" : std::cout << "\nPolynomials are not equal\n\n";

Polynomial\* pp1 = new Polynomial;

Polynomial\* pp2 = new Polynomial;

pp1->push(4, datas1);

pp2->push(4, datas0);

std::cout << "Does pp1 and pp2 are not equal?";

\*pp1 != \*pp2 ? std::cout << "\nPolynomials are not equal\n\n" : std::cout << "\nPolynomials are equal!!!\n\n";

std::cout << "\n\n";

std::cout << \*pol1;

std::cout << "when x = 0, pol1 = " << (\*pol1)(0.) << "\n";

std::cout << "when x = 1, pol1 = " << (\*pol1)(1.) << "\n";

std::cout << "when x = -1.01, pol1 = " << (\*pol1)(-1.01) << "\n\n";

/\*Polynomial\* pp3 = new Polynomial;

std::cout << (\*pp3)(-1.) << "\n";\*/

std::cout << "\n\n";

std::cout << "pol1[0] = " << (\*pol1)[0] << "\n";

std::cout << "pol1[1] = " << (\*pol1)[1] << "\n";

std::cout << "pol1[2] = " << (\*pol1)[2] << "\n";

std::cout << "pol1[3] = " << (\*pol1)[3] << "\n\n";

/\*std::cout << (\*pol1)[4] << "\n";

std::cout << (\*pol1)[5] << "\n";\*/

/\*Polynomial\* pp3 = new Polynomial;

std::cout << (\*pp3)[1] << "\n";\*/

std::cout << "\nPOINT 1\n";

Polynomial\* pp3 = new Polynomial;

Polynomial\* pp4 = new Polynomial;

\*pp4 = \*pp3 + \*data0;

\*pp4 = \*pp4 + \*data1;

\*pp4 = \*data0 + \*pp4;

\*pp4 = \*pp4 + \*pp4;

\*pp4 = \*pp4 + \*pp4;

\*pp4 = \*pp4 + \*pp4;

std::cout << \*pp4;

std::cout << "\nPOINT 2\n";

Polynomial\* pp5 = new Polynomial;

\*pp5 = \*pp5 - \*pp5;

std::cout << \*pp5;

std::cout << "\nPOINT 3\n";

Polynomial\* pp6 = new Polynomial;

std::cout << \*pol1;

\*pp6 = \*pol1;

std::cout << \*pp6;

\*pp6 = \*pp6 \* 3.001;

std::cout << \*pp6;

\*pp6 = 0 \* \*pp6;

std::cout << \*pp6;

\*pp6 = 0 \* \*pp6;

std::cout << \*pp6;

std::cout << "\nPOINT 4\n";

Polynomial\_info\* data01 = new Polynomial\_info(0, 0);

Polynomial\_info\* data02 = new Polynomial\_info(0, 1.5);

Polynomial\_info\* data03 = new Polynomial\_info(0, 0);

Polynomial\_info\* datas00[3] = { data01, data02, data03 };

Polynomial\* pol4 = new Polynomial(3, datas00);

std::cout << \*pol4;

std::cout << "\nPOINT 5\n";

std::cout << \*pp6;

std::cout << \*pol4;

std::cout << "\nPOINT 6\n";

\*pp6 = std::move(\*pol4);

std::cout << \*pp6;

std::cout << \*pol4;

std::cout << "\nPOINT 7\n";

delete pol1;

Polynomial pp7(std::move(\*pp6));

std::cout << \*pp6;

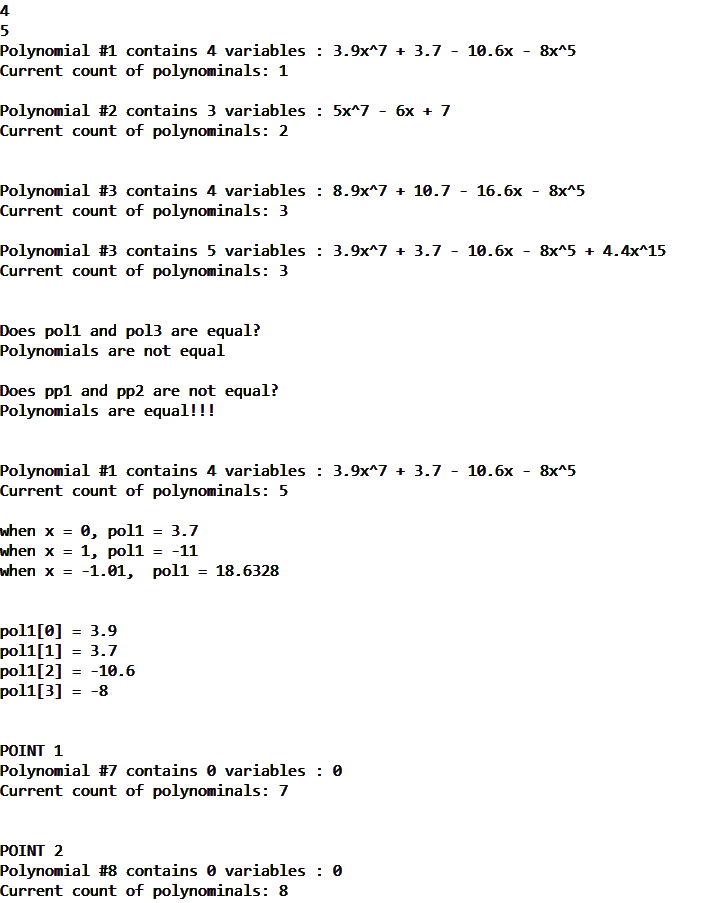
std::cout << "\nPOINT 8\n";

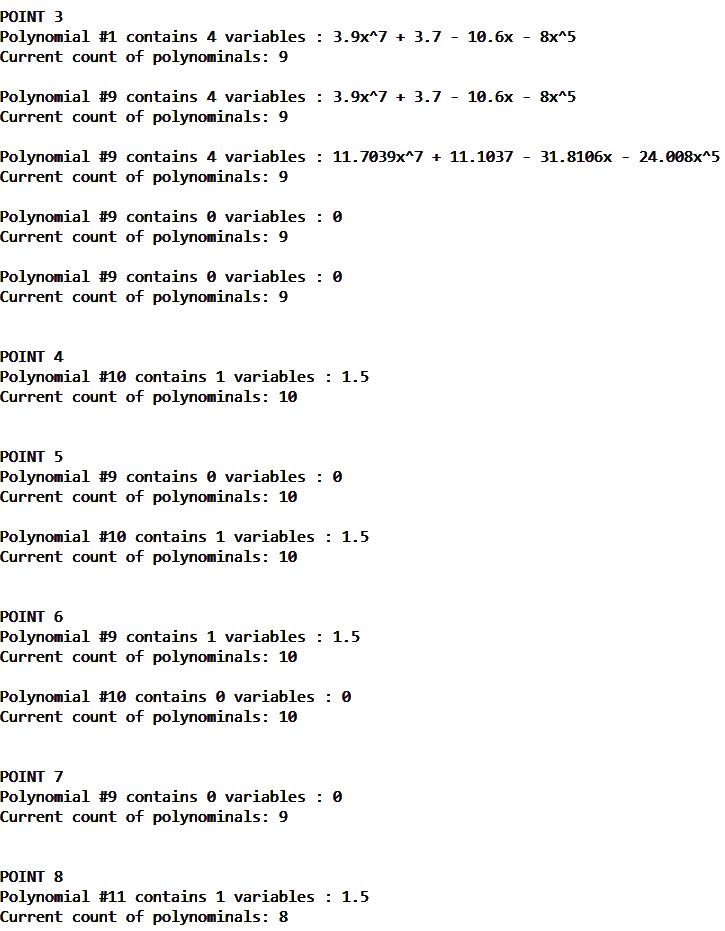
delete pol2;

std::cout << pp7;

return 0;

}

**ВИКОНАННЯ ЕКСПЕРИМЕНТІВ ПО НАЛАГОДЖЕННЮ ПРОГРАМИ **

****

**Висновки.**

Протягом лабораторної роботи мною було засвоєнно поняття перевантаження операцій, були засвоєні навичоки перевантаження унарних та бінарних операцій методами класу та за допомогою дружніх функцій.