МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Інститут комп'ютерних наук та інформаційних технологій
Кафедра програмного забезпечення

ЗВІТ

До лабораторної роботи №1
На тему: «Створення та використання класів»
З дисципліни: «Об'єктно-орієнтоване програмування»

Лектор:
доц. кафедри ПЗ
Коротєєва Т. О.

Виконав:
ст. гр. ПЗ-16
Чаус О. М.

Прийняв:
асист. Дивак І. В.

« ____ » _____ 2022 р.

∑= _____ .....:

Львів-2022

Тема роботи: Створення та використання класів

Мета роботи: Навчитися створювати класи, використовувати конструктори для ініціалізації об’єктів, опанувати принципи створення функцій-членів. Навчитися використовувати різні типи доступу до полів та методів класів.

Теоретичні відомості

Клас є типом даних, який визначається користувачем. У класі задаються властивості і поведінка будь-якого предмету або процесу у вигляді полів даних (аналогічно до того як це є в структурах) і функцій для роботи з ними. Створюваний тип даних володіє практично тими ж властивостями, що і стандартні типи.

Опис класу в першому наближенні виглядає так:

```
class <ім'я> {
[private:]
<Опис прихованих елементів>
public:
<Опис доступних елементів>
}; //Опис закінчується крапкою з комою.
```

Специфікатор доступу private і public керують видимістю елементів класу. Елементи, описані після службового слова private, видимі тільки всередині класу. Цей вид доступу прийнятий у класі за замовчуванням. Інтерфейс класу описується після специфікатора public. Дія будь-якого специфікатора поширюється до наступного специфікатора або до кінця класу. Можна задавати кілька секцій private і public, їх порядок значення не має.

Поля класу:

• можуть мати будь-який тип, крім типу цього ж класу (але можуть бути вказівниками або посиланнями на цей клас);

• можуть бути описані з модифікатором const, при цьому вони ініціалізуються тільки один раз (за допомогою конструктора) і не можуть змінюватися;

• можуть бути описані з модифікатором static (розглядається в наступних лабораторних). Ініціалізація полів при описі не допускається.

Конструктори

Конструктор призначений для ініціалізації об’єкту і викликається автоматично при його створенні. Автоматичний виклик конструктора дозволяє уникнути помилок, пов’язаних з використанням неініціалізованих змінних. Нижче наведені основні властивості конструкторів:

• Конструктор не повертає жодного значення, навіть типу void. Неможливо отримати вказівник на конструктор.

• Клас може мати декілька конструкторів з різними параметрами для різних видів ініціалізації (при цьому використовується механізм перевантаження).

• Конструктор без параметрів називається конструктором за замовчуванням.

• Параметри конструктора можуть мати будь-який тип, крім цього ж класу. Можна задавати значення параметрів за замовчуванням. Їх може містити тільки один з конструкторів.

• Якщо програміст не вказав жодного конструктора, компілятор створює його автоматично. Такий конструктор викликає конструктори за замовчуванням для полів класу і конструктори за замовчуванням базових класів. У разі, коли клас містить константи або посилання, при спробі створення об'єкту класу буде видана помилка, оскільки їх необхідно ініціалізувати конкретними значеннями, а конструктор за замовчуванням цього робити не вміє.

• Конструктори не наслідуються.

• Конструктори не можна описувати з модифікаторами const, virtual і static.

• Конструктори глобальних об’єктів викликаються до виклику функції main. Локальні об’єкти створюються, як тільки стає активною область їх дії. Конструктор запускається і при створенні тимчасового об'єкта (наприклад, при передачі об’єкта з функції).
• Конструктор викликається, якщо в програмі зустрілася будь-яка із синтаксичних конструкцій:
імя_класу ім’я_об’єкту [(список параметрів)];
//Список параметрів не повинен бути порожнім
імя_класу (список параметрів);
//Створюється об'єкт без імені (список може бути //порожнім)
ім’я_класу ім’я_об’єкту = вираз;
//Створюється об’єкт без імені і копіюється

Завдання для лабораторної роботи

1. Створити клас відповідно до варіанту (див. Додаток).
   2. При створенні класу повинен бути дотриманий принцип інкапсуляції.
   3. Створити конструктор за замовчуванням та хоча б два інших конструктори для початкової ініціалізації об’єкта.
   4. Створити функції члени згідно з варіантом.
   5. Продемонструвати можливості класу завдяки створеному віконному застосуванню.
   6. У звіті до лабораторної намалювати UML-діаграму класу, яка відповідає варіанту.

   Клас **Polynom** – квадратичний тричлен ( ). Клас повинен містити функції-члени, які реалізовують: а)Знаходження значення виразу для заданого . б)Знаходження значення похідної в заданій тч. в)Знаходження визначеного інтегралу на заданому проміжку г)Знаходження коренів рівняння д)Додавання двох поліномів е)Задавання значень полів є)Зчитування (отримання значень полів) ж)Множення полінома на число з)Введення полінома з форми и)Виведення полінома на форму.

Хід роботи

Файл class.h:

```
#pragma once
struct Roots
{
	double x1;
	double x2;
	bool solveable;
};
ref class a
{

};
ref class Polynom
{
private:
	double m_a;
	double m_b;
	double m_c;
public:
	Polynom()
	{
		m_a = 0;
		m_b = 0;
		m_c = 0;
	}
	Polynom(double a, double b, double c)
	{
```

```
		m_a = a;
		m_b = b;
		m_c = c;
	}
	Polynom(double* array)
	{
		m_a = array[0];
		m_b = array[1];
		m_c = array[2];
	}
	Polynom(Polynom% A)
	{
		m_a = A.m_a;
		m_b = A.m_b;
		m_c = A.m_c;
	}
	double Result(double x);
	double Derivarive(double point);
	double Integral(double low, double high);
	Roots findRoots();
	void add(Polynom poly);
	void multiply(double mult);
	void readForm(System::Windows::Forms::TextBox^ A, System::Windows::Forms::TextBox^ B,
System::Windows::Forms::TextBox^ C);
	void outputForm(System::Windows::Forms::TextBox^ A, System::Windows::Forms::TextBox^
B, System::Windows::Forms::TextBox^ C);
	double getA();
	double getB();
	double getC();
	void   setA(double a);
	void   setB(double b);
	void   setC(double c);
};
```

Файл class.cpp:

```
#include "class.h"
#include <cmath>

double Polynom::Result(double x)
{
	return m_a * x * x + m_b * x + m_c;
}
double Polynom::Derivarive(double point)
{
	return 2 * m_a * point + m_b;
}
double Polynom::Integral(double low, double high)
{
	return ((m_a * high * high * high) / 3 + (m_b * high * high) / 2 + m_c * high) -
		((m_a * low * low * low) / 3 + (m_b * low * low) / 2 + m_c * low);
}
Roots Polynom::findRoots()
{
	double D = m_b * m_b - 4 * m_a * m_c;
	if (D < 0)
	{
		return { 0, 0, false };
	}
	else
	{
		double x1 = (-1 * m_b - sqrt(D)) / (2 * m_a);
		double x2 = (-1 * m_b + sqrt(D)) / (2 * m_a);
		return { x1, x2, true };
	}
}
void Polynom::readForm(System::Windows::Forms::TextBox^ A, System::Windows::Forms::TextBox^
B, System::Windows::Forms::TextBox^ C)
{
```

```
	m_a = 0;
	m_b = 0;
	m_c = 0;
	if (A->Text != "")
	{
		m_a = System::Convert::ToDouble(A->Text);
	}
	if (B->Text != "")
	{
		m_b = System::Convert::ToDouble(B->Text);
	}
	if (C->Text != "")
	{
		m_c = System::Convert::ToDouble(C->Text);
	}
}
void Polynom::outputForm(System::Windows::Forms::TextBox^ A,
System::Windows::Forms::TextBox^ B, System::Windows::Forms::TextBox^ C)
{
	A->Text = System::Convert::ToString(m_a);
	B->Text = System::Convert::ToString(m_b);
	C->Text = System::Convert::ToString(m_c);
}
double Polynom::getA()
{
	return m_a;
}
double Polynom::getB()
{
	return m_b;
}
double Polynom::getC()
{
	return m_c;
}
void Polynom::add(Polynom poly)
{
	double a = poly.getA();
	m_a += a;
	double b = poly.getB();
	m_b += b;
	double c = poly.getC();
	m_c += c;
}
void Polynom::multiply(double mult)
{
	m_a *= mult;
	m_b *= mult;
	m_c *= mult;
}
void	Polynom::setA(double a)
{
	m_a = a;
}
void	Polynom::setB(double b)
{
	m_b = b;
}
void	Polynom::setC(double c)
{
	m_c = c;
}
```

Файл form.h:

```
#include "class.h"

		bool resClicked, derivativeClicked, integralClicked, plusClicked,
multiplyClicked;
```

```
		Polynom poly;
private: System::Void buttonRes_Click(System::Object^ sender, System::EventArgs^ e) {
	resClicked = true;
	derivativeClicked = false;
	integralClicked = false;
	plusClicked = false;
	multiplyClicked = false;
	textBoxX->ReadOnly = false;
	textBox1->ReadOnly = true;
	textBox2->ReadOnly = true;
	buttonEqual->Enabled = true;
}
private: System::Void buttonDer_Click(System::Object^ sender, System::EventArgs^ e) {
	resClicked = false;
	derivativeClicked = true;
	integralClicked = false;
	multiplyClicked = false;
	plusClicked = false;
	textBoxX->ReadOnly = false;
	textBox1->ReadOnly = true;
	textBox2->ReadOnly = true;
	buttonEqual->Enabled = true;
}
private: System::Void buttonIntegral_Click(System::Object^ sender, System::EventArgs^ e) {
	resClicked = false;
	derivativeClicked = false;
	multiplyClicked = false;
	integralClicked = true;
	plusClicked = false;
	textBoxX->ReadOnly = false;
	textBox1->ReadOnly = false;
	textBox2->ReadOnly = false;
	buttonEqual->Enabled = true;
}
private: System::Void buttonPlus_Click(System::Object^ sender, System::EventArgs^ e) {
	resClicked = false;
	derivativeClicked = false;
	integralClicked = false;
	multiplyClicked = false;
	plusClicked = true;
	textBoxX->ReadOnly = true;
	textBox1->ReadOnly = true;
	textBox2->ReadOnly = true;
	buttonEqual->Enabled = true;
	poly.readForm(textBoxA, textBoxB, textBoxC);
	textBoxA->Clear();
	textBoxB->Clear();
	textBoxC->Clear();
}
private: System::Void buttonMultiply_Click(System::Object^ sender, System::EventArgs^ e) {
	resClicked = false;
	derivativeClicked = false;
	integralClicked = false;
	plusClicked = false;
	multiplyClicked = true;
	textBoxX->ReadOnly = false;
	textBox1->ReadOnly = true;
	textBox2->ReadOnly = true;
	buttonEqual->Enabled = true;
}
private: System::Void buttonRoots_Click(System::Object^ sender, System::EventArgs^ e) {
	poly.readForm(textBoxA, textBoxB, textBoxC);
	Roots roots = poly.findRoots();
	if (roots.solveable)
	{
		if (roots.x1 == roots.x2)
		{
			MessageBox::Show("X = " + roots.x1);
```

```
			}
			else
				MessageBox::Show("X1 = " + roots.x1 + ", X2 = " + roots.x2);
		}
		else MessageBox::Show("No roots");
	}
	private: System::Void buttonEqual_Click(System::Object^ sender, System::EventArgs^ e) {
		if (plusClicked)
		{
			Polynom poly2;
			poly2.readForm(textBoxA, textBoxB, textBoxC);
			poly.add(poly2);
			poly.outputForm(textBoxA, textBoxB, textBoxC);
			buttonEqual->Enabled = false;
		}
		else
		{
			poly.readForm(textBoxA, textBoxB, textBoxC);
		}
		if (multiplyClicked)
		{
			if (textBoxX->Text != "")
			{
				poly.multiply(Double::Parse(textBoxX->Text));
				poly.outputForm(textBoxA, textBoxB, textBoxC);
				multiplyClicked = false;
				textBoxX->Text = "";
				textBoxX->ReadOnly = true;
				buttonEqual->Enabled = false;
			}
		}
		if (resClicked)
		{
			if (textBoxX->Text != "")
			{
				Result->Text = (poly.Result(Double::Parse(textBoxX->Text)).ToString());
				resClicked = false;
				textBoxX->Text = "";
				textBoxX->ReadOnly = true;
				buttonEqual->Enabled = false;
			}
		}
		if (derivativeClicked)
		{
			if (textBoxX->Text != "")
			{
				Result->Text = (poly.Derivarive(Double::Parse(textBoxX-
>Text)).ToString());
				derivativeClicked = false;
				textBoxX->Text = "";
				textBoxX->ReadOnly = true;
				buttonEqual->Enabled = false;
			}
		}
		if (integralClicked)
		{
			if (textBox1->Text != "" && textBox2->Text != "")
			{
				Result->Text = (poly.Integral(Double::Parse(textBox1->Text),
Double::Parse(textBox2->Text))).ToString();

				integralClicked = false;
				textBox1->Text = "";
				textBox2->Text = "";
				textBox1->ReadOnly = true;
				textBox2->ReadOnly = true;
				buttonEqual->Enabled = false;
			}
```

```
	}
}
private: System::Void textBoxX_KeyPress(System::Object^ sender,
System::Windows::Forms::KeyPressEventArgs^ e) {
	if (!Char::IsControl(e->KeyChar) && !Char::IsDigit(e->KeyChar) && e->KeyChar != ',')
	{
		e->Handled = true;
	}
	if (e->KeyChar == ',' && textBoxX->Text->Contains(","))
	{
		e->Handled = true;
	}
}
private: System::Void textBox1_KeyPress(System::Object^ sender,
System::Windows::Forms::KeyPressEventArgs^ e) {
	if (!Char::IsControl(e->KeyChar) && !Char::IsDigit(e->KeyChar) && e->KeyChar != ',')
	{
		e->Handled = true;
	}
	if (e->KeyChar == ',' && textBox1->Text->Contains(","))
	{
		e->Handled = true;
	}
}
private: System::Void textBox2_KeyPress(System::Object^ sender,
System::Windows::Forms::KeyPressEventArgs^ e) {
	if (!Char::IsControl(e->KeyChar) && !Char::IsDigit(e->KeyChar) && e->KeyChar != ',')
	{
		e->Handled = true;
	}
	if (e->KeyChar == ',' && textBox2->Text->Contains(","))
	{
		e->Handled = true;
	}
}
private: System::Void textBoxA_KeyPress(System::Object^ sender,
System::Windows::Forms::KeyPressEventArgs^ e) {
	if (!Char::IsControl(e->KeyChar) && !Char::IsDigit(e->KeyChar) && e->KeyChar != ',')
	{
		e->Handled = true;
	}
	if (e->KeyChar == ',' && textBoxA->Text->Contains(","))
	{
		e->Handled = true;
	}
}
private: System::Void textBoxB_KeyPress(System::Object^ sender,
System::Windows::Forms::KeyPressEventArgs^ e) {
	if (!Char::IsControl(e->KeyChar) && !Char::IsDigit(e->KeyChar) && e->KeyChar != ',')
	{
		e->Handled = true;
	}
	if (e->KeyChar == ',' && textBoxB->Text->Contains(","))
	{
		e->Handled = true;
	}
}
private: System::Void textBoxC_KeyPress(System::Object^ sender,
System::Windows::Forms::KeyPressEventArgs^ e) {
	if (!Char::IsControl(e->KeyChar) && !Char::IsDigit(e->KeyChar) && e->KeyChar != ',')
	{
		e->Handled = true;
	}
	if (e->KeyChar == ',' && textBoxC->Text->Contains(","))
	{
		e->Handled = true;
	}
}
```

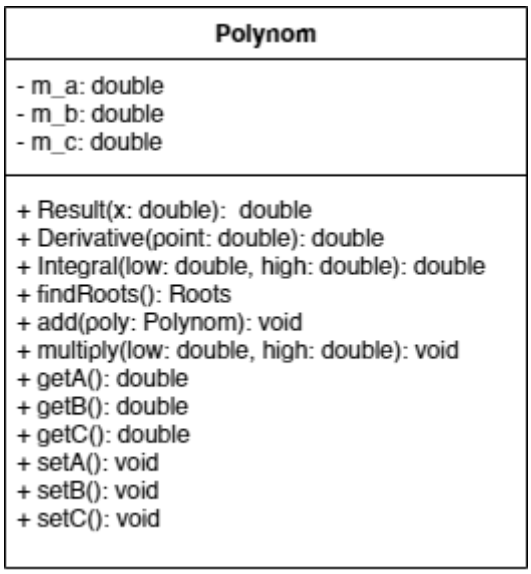
```
};
}
```

Файл main.cpp:

```
#include "form.h"

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

[STAThreadAttribute]
void Main(array<String^>^ args) {
    Application::EnableVisualStyles();
    Application::SetCompatibleTextRenderingDefault(false);
    OOPLab05::form form;
    Application::Run(% form);
}
```



Діаграма класу Polynom

Скріншоти

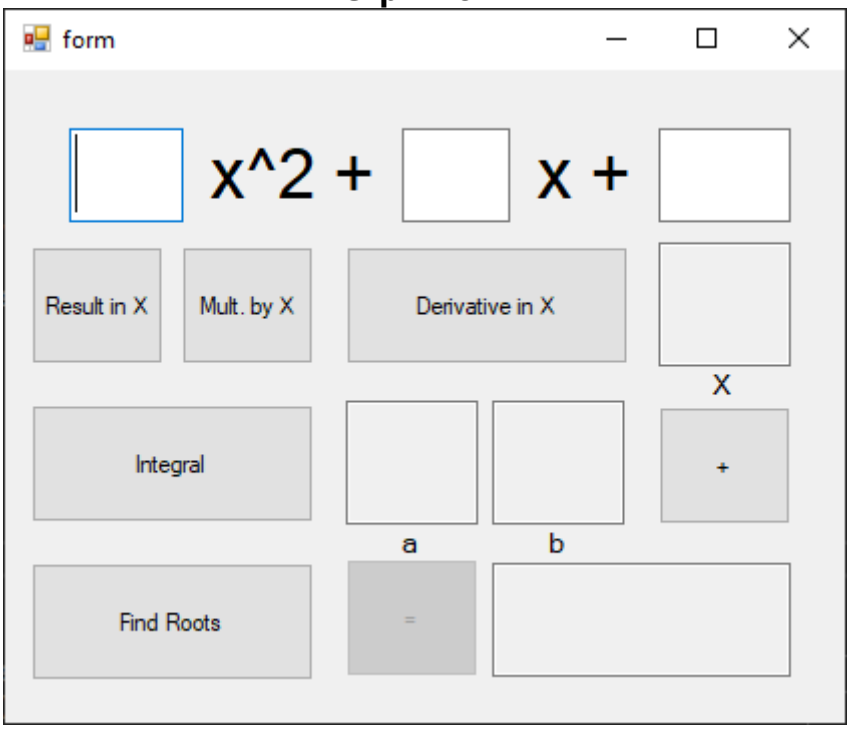


Рис. 1. Програма при запуску

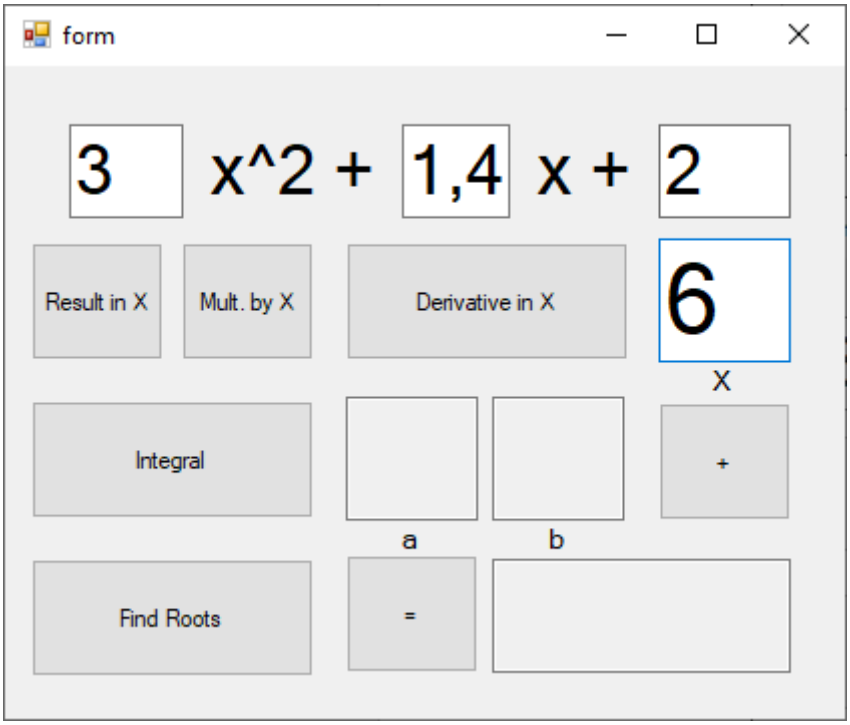


Рис. 2. Введення даних для знаходження значення виразу

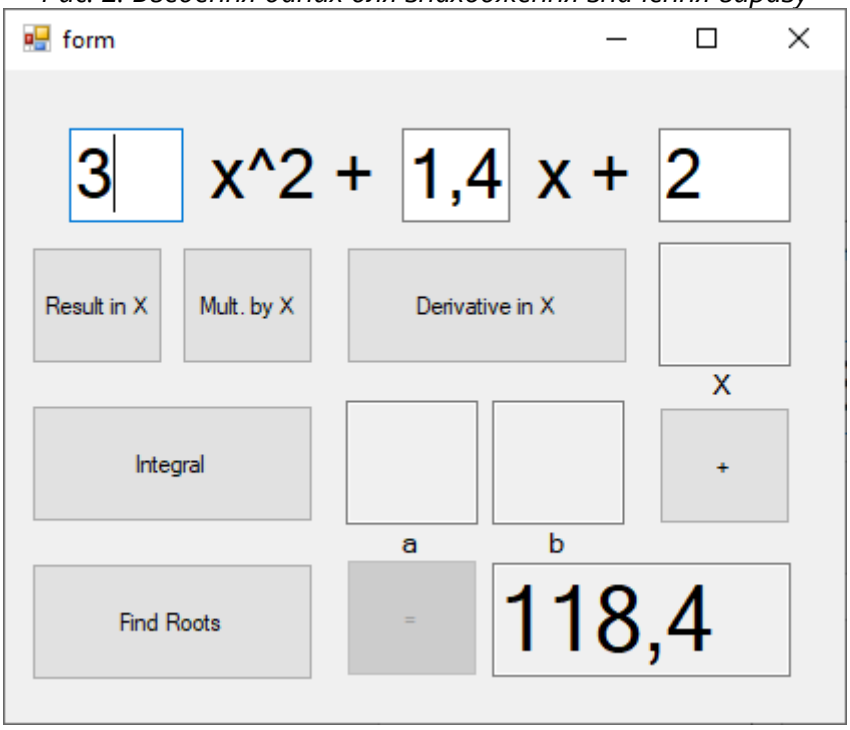
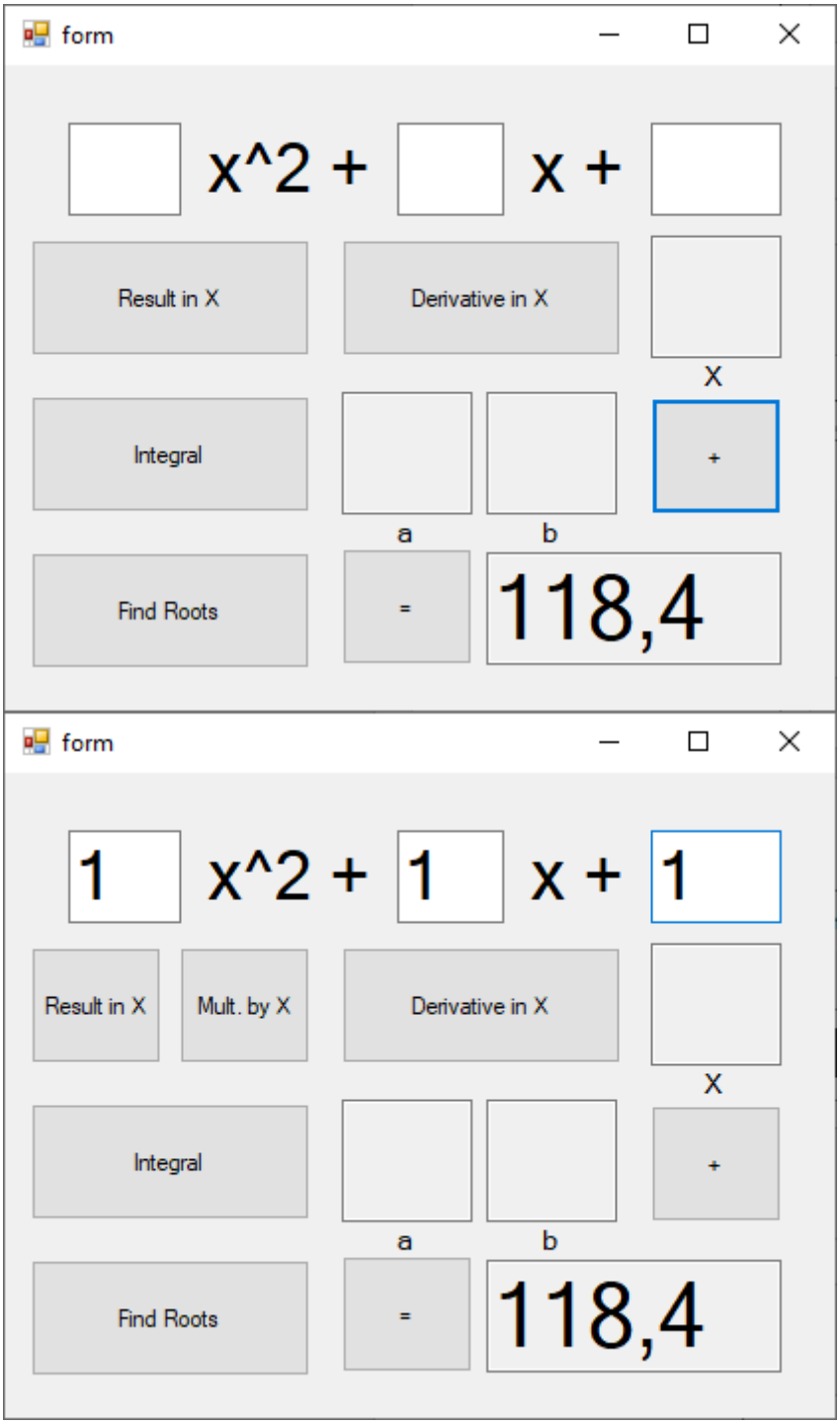


Рис. 3. Результат обчислення


form
x^2 +
x +
Result in X
Derivative in X
X
Integral
+
a
b
Find Roots
=
118,4
form
1
x^2 +
1
x +
1
Result in X
Mult. by X
Derivative in X
X
Integral
+
a
b
Find Roots
=
118,4

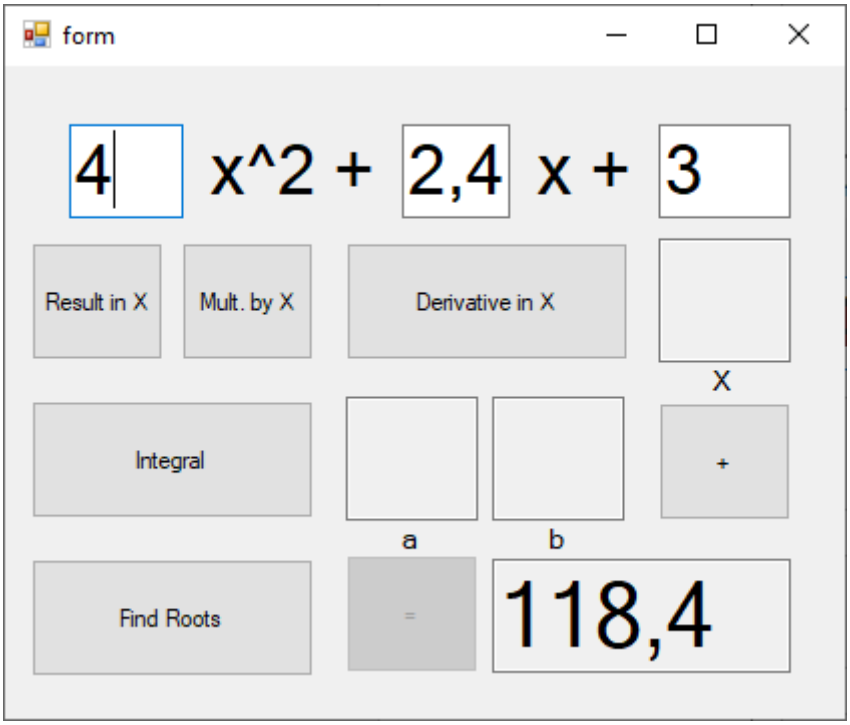


Рис. 4-6. Додавання двох квадратичних тричленів

Висновок: навчився створювати класи, використовувати конструктори для ініціалізації об’єктів, опанував принципи створення функцій-членів. Навчився використовувати різні типи доступу до полів та методів класів.