Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Інститут комп’ютерних наук та інформаційних технологій

Кафедра програмного забезпечення



**Звіт**

Про виконання лабораторної роботи №4

**На тему:**

“Перевантаження функцій і операторів, дружні функції, статичні члени класу”

з дисципліни “Об’єктно-орієнтоване програмування”

**Лектор:**

доц. каф. ПЗ

Коротєєва Т.О.

**Виконав:**

ст. гр. ПЗ-18

Юшкевич А.І.

**Прийняв:**

асис. каф. ПЗ

Дивак І.В.

« … » … 2023 р.

∑ = \_\_\_\_\_\_\_\_\_\_\_\_

Львів – 2023

**Тема**: Перевантаження функцій і операторів, дружні функції, статичні члени класу.

**Мета**: Навчитися використовувати механізм перевантаження функцій та операторів. Навчитися створювати та використовувати дружні функції. Ознайомитися зі статичними полями і методами та навчитися їх використовувати.

Теоретичні відомості

**Перевизначення операторів**

С++ підтримує спеціальні засоби, які дозволяють перевизначити вже існуючі оператори. Наприклад, для оператора + можна ввести своє власне визначення, яке реалізує операцію додавання для об’єктів певного класу. Фактично перевизначення для операцій існує і в мові С. Так, операція + може використовувати як об’єкти типу int, так і об’єкти типу float. С++ розширює цю ідею.

Для визначення оператора використовується функція, що вводиться користувачем. Тип функції визначається іменем класу, далі записується ключове слово operator, за яким слідує назва оператора, в круглих дужках дається перелік параметрів, серед яких хоча б один типу клас.

**Дружні функції**

Дружньою функцією класу називається функція, яка сама не є членом класу, але має повні права на доступ до закритих та захищених елементів класу. Оскільки така функція не є членом класу, то вона не може бути вибрана з допомогою операторів (.) та (->). Дружня функція класу викликається звичайним способом. Для опису дружньої функції використовується ключове слово friend.

**Статичні змінні класу**

До тепер вважалось, що дані в кожному об’єкті є власністю саме цього об’єкту та не використовуються іншими об’єктами цього класу. Та іноді приходиться слідкувати за накопиченням даних. Наприклад, необхідно з’ясувати скільки об’єктів даного класу було створено на даний момент та скільки з них існує. Статичні змінні-члени досяжні для всіх екземплярів класу. Це є компроміс між глобальними даними, які досяжні всім елементам програми, та даними, що досяжні тільки об’єктам даного класу.

# Завдання

На основі класу з попередньої лабораторної: • Перевантажити як мінімум три функції-члени з попереднього завдання.

* Перевантажити оператори згідно з варіантом.
* Створити дружні функції згідно з варіантом.
* Створити статичні поля та статичні методи згідно з варіантом.
* Створити статичне поле, в якому б містилась інформація про кількість створених об’єктів, а також статичні функції для роботи з цим полем.
* Продемонструвати можливості класу написавши для нього модульні тести для Google Test.
* Оформити звіт до лабораторної роботи. Звіт повинен містити:
  + Тему, мету та теоретичні відомості
  + Індивідуальне завдання
  + Код класу
  + Код тестів
  + Результат виконання тестів

Варіант №12:

|  |  |
| --- | --- |
| 12 | Клас СPolynom4 – кубічний чотиричлен (ax3+bx2+cx+d).  Перевантажити оператори, як функції члени:   * Додавання (+) * Віднімання (-) * Знаходження значення виразу для заданого x ( () ) * Заміна всіх коефіцієнтів полінома на протилежні за знаком (!) Перевантажити оператори, як дружні-функції: * Добуток полінома на скаляр * Введення полінома з потоку вводу (<<) * Виведення полінома в потік виводу (>>) * Доступ до і-го коефіцієнта полінома ( [] ) * Рівне (==) (при порівнянні порівнювати значення всіх коефіцієнтів). |

# Код програми

## Lab\_04\_OOP:

Lab\_04\_OOP.cpp:

#include <iostream>

#include "Polynom\_Header.h"

int CPolynom4::number\_of\_objects{ 0 };

CPolynom4::CPolynom4(double a, double b, double c, double d) {

number\_of\_objects++;

polynom.a = a;

polynom.b = b;

polynom.c = c;

polynom.d = d;

}

CPolynom4::CPolynom4(SPolynom polynom) {

number\_of\_objects++;

this->polynom = polynom;

}

CPolynom4::CPolynom4() {

number\_of\_objects++;

this->polynom.a = 1;

this->polynom.b = 1;

this->polynom.c = 1;

this->polynom.d = 1;

}

double CPolynom4::GetResult(double x) {

return polynom.a \* x \* x \* x + polynom.b \* x \* x + polynom.c \* x + polynom.d;

}

double CPolynom4::GetDerivative(double x) {

return 3 \* polynom.a \* x \* x + 2 \* polynom.b \* x + polynom.c;

}

double CPolynom4::GetIntegral(double x1, double x2) {

return polynom.a \* x2 \* x2 \* x2 \* x2 / 4 + polynom.b \* x2 \* x2 \* x2 / 3 + polynom.c \* x2 \* x2 / 2 + polynom.d \* x2

- (polynom.a \* x1 \* x1 \* x1 \* x1 / 4 + polynom.b \* x1 \* x1 \* x1 / 3 + polynom.c \* x1 \* x1 / 2 + polynom.d \* x1);

}

SResult CPolynom4::GetRoots() {

RootsFinder finder(polynom);

return finder.FindByDichotomy();

}

SResult CPolynom4::GetRoots(ldouble leftLim, ldouble rightLim) {

RootsFinder finder(leftLim, rightLim, polynom);

return finder.FindByDichotomy();

}

SResult CPolynom4::GetRoots(ldouble leftLim, ldouble rightLim, ldouble epsilon) {

RootsFinder finder(leftLim, rightLim, epsilon, polynom);

return finder.FindByDichotomy();

}

SPolynom CPolynom4::GetPolynom() {

return polynom;

}

void CPolynom4::SetPolynom(ldouble a, ldouble b, ldouble c, ldouble d) {

polynom.a = a;

polynom.b = b;

polynom.c = c;

polynom.d = d;

}

void CPolynom4::SetPolynom(SPolynom polynom) {

this->polynom = polynom;

}

CPolynom4 CPolynom4::operator + (const CPolynom4& other) {

double a, b, c, d;

a = this->polynom.a + other.polynom.a;

b = this->polynom.b + other.polynom.b;

c = this->polynom.c + other.polynom.c;

d = this->polynom.d + other.polynom.d;

CPolynom4 NewPol(a, b, c, d);

return NewPol;

}

CPolynom4 CPolynom4::operator - (const CPolynom4& other) {

double a, b, c, d;

a = this->polynom.a - other.polynom.a;

b = this->polynom.b - other.polynom.b;

c = this->polynom.c - other.polynom.c;

d = this->polynom.d - other.polynom.d;

CPolynom4 NewPol(a, b, c, d);

return NewPol;

}

bool operator == (const CPolynom4& superior, const CPolynom4& other) {

if ( superior.polynom.a == other.polynom.a

&& superior.polynom.b == other.polynom.b

&& superior.polynom.c == other.polynom.c

&& superior.polynom.d == other.polynom.d

)

{

return true;

}

else

return false;

}

bool operator != (const CPolynom4& superior, const CPolynom4& other) {

if ( superior.polynom.a != other.polynom.a

|| superior.polynom.b != other.polynom.b

|| superior.polynom.c != other.polynom.c

|| superior.polynom.d != other.polynom.d

)

{

return true;

}

else

return false;

}

ldouble CPolynom4::operator()(const ldouble x) {

return GetResult(x);

}

CPolynom4 CPolynom4::operator!() {

return CPolynom4(-this->polynom.a, -this->polynom.b, -this->polynom.c, -this->polynom.d);

}

CPolynom4 operator\* (const CPolynom4& superior, const ldouble scalar) {

double a, b, c, d;

a = superior.polynom.a \* scalar;

b = superior.polynom.b \* scalar;

c = superior.polynom.c \* scalar;

d = superior.polynom.d \* scalar;

CPolynom4 NewPol(a, b, c, d);

return NewPol;

}

ostream& operator << (ostream& os, const CPolynom4& superior) {

os << endl << endl << "(" << superior.polynom.a << ")x^3 + (" << superior.polynom.b << ")x^2 + (" << superior.polynom.c << ")x + (" << superior.polynom.d << ")" << endl << endl;

return os;

}

istream& operator >> (istream& is, CPolynom4& superior) {

is >> superior.polynom.a >> superior.polynom.b >> superior.polynom.c >> superior.polynom.d;

return is;

}

ldouble& CPolynom4::operator [](int index) {

while (true) {

switch (index)

{

case 0:

return this->polynom.a;

break;

case 1:

return this->polynom.b;

break;

case 2:

return this->polynom.c;

break;

case 3:

return this->polynom.d;

break;

default:

if (index < 0)

index += 4;

else

index -= 4;

break;

}

}

}

### Polynom\_Header.h:

#pragma once

#include <iostream>

#include "RootsHeader.h"

class CPolynom4 {

private:

static int number\_of\_objects;

SPolynom polynom;

public:

CPolynom4(double a, double b, double c, double d);

CPolynom4(SPolynom polynom);

CPolynom4();

double GetResult(double x);

double GetDerivative(double x);

double GetIntegral(double x1, double x2);

SResult GetRoots();

SResult GetRoots(ldouble leftLim, ldouble rightLim);

SResult GetRoots(ldouble leftLim, ldouble rightLim, ldouble epsilon);

SPolynom GetPolynom();

void SetPolynom(ldouble a, ldouble b, ldouble c, ldouble d);

void SetPolynom(SPolynom polynom);

static int GetNumOfObjects() {

return number\_of\_objects;

}

CPolynom4 operator + (const CPolynom4& other);

CPolynom4 operator - (const CPolynom4& other);

ldouble operator () (const ldouble x);

CPolynom4 operator ! ();

ldouble& operator [] (int index);

friend ostream& operator << (ostream& os, const CPolynom4& superior);

friend istream& operator >> (istream& is, CPolynom4& superior);

friend CPolynom4 operator \* (const CPolynom4& superior, const ldouble scalar);

friend bool operator == (const CPolynom4& superior, const CPolynom4& other);

friend bool operator != (const CPolynom4& superior, const CPolynom4& other);

};

## Methods\_Test:

### pch.h:

#pragma once

#include "gtest/gtest.h"

#include "..\Lab\_04\_OOP\Polynom\_Header.h"

### test.cpp:

#include "pch.h"

SPolynom pol\_1{ 2128.353, 9354.3453, 63568.72257, 385561.3747 },

pol\_2{ 73427.8745, 5674.12087, 5367.54428, 584.2626 },

pol\_3{ 0, 0, 0, 0 },

pol\_4{ -2346.2346, -2087.5473, -2326.5428, -23873.45372 },

pol\_5{ 1'000'000'000'000, 1'000'000'000'000, 1'000'000'000'000, 1'000'000'000'000 };

TEST(CPolynom4Test, Number\_of\_objects) {

CPolynom4 p1(pol\_1);

CPolynom4 p2(pol\_2);

CPolynom4 p3(pol\_3);

CPolynom4 p4(pol\_4);

int number\_of\_objects = CPolynom4::GetNumOfObjects();

EXPECT\_EQ(number\_of\_objects, 4);

}

TEST(Plus, HandlesNormalInput) {

CPolynom4 C1(pol\_1), C2(pol\_2), Sum;

Sum = C1 + C2;

SPolynom result = Sum.GetPolynom();

EXPECT\_DOUBLE\_EQ(result.a, 75'556.2275);

EXPECT\_DOUBLE\_EQ(result.b, 15'028.46617);

EXPECT\_DOUBLE\_EQ(result.c, 68'936.26685);

EXPECT\_DOUBLE\_EQ(result.d, 386'145.6373);

}

TEST(Plus, HandlesZeroInput) {

CPolynom4 C1(pol\_3), C2(pol\_2), Sum;

Sum = C1 + C2;

SPolynom result = Sum.GetPolynom();

EXPECT\_DOUBLE\_EQ(result.a, pol\_2.a);

EXPECT\_DOUBLE\_EQ(result.b, pol\_2.b);

EXPECT\_DOUBLE\_EQ(result.c, pol\_2.c);

EXPECT\_DOUBLE\_EQ(result.d, pol\_2.d);

}

TEST(Plus, HandlesBigInput) {

CPolynom4 C1(pol\_5), C2(pol\_5), Sum;

Sum = C1 + C2;

SPolynom result = Sum.GetPolynom();

EXPECT\_DOUBLE\_EQ(result.a, 2'000'000'000'000);

EXPECT\_DOUBLE\_EQ(result.b, 2'000'000'000'000);

EXPECT\_DOUBLE\_EQ(result.c, 2'000'000'000'000);

EXPECT\_DOUBLE\_EQ(result.d, 2'000'000'000'000);

}

TEST(Plus, HandlesNegativeInput) {

CPolynom4 C1(pol\_1), C2(pol\_4), Sum;

Sum = C1 + C2;

SPolynom result = Sum.GetPolynom();

EXPECT\_DOUBLE\_EQ(result.a, 2128.353 - 2346.2346);

EXPECT\_DOUBLE\_EQ(result.b, 7'266.798);

EXPECT\_DOUBLE\_EQ(result.c, 61'242.17977);

EXPECT\_DOUBLE\_EQ(result.d, 361'687.92098);

}

TEST(Minus, HandlesNormalInput) {

CPolynom4 C1(pol\_1), C2(pol\_2), Diff;

Diff = C1 - C2;

SPolynom result = Diff.GetPolynom();

EXPECT\_DOUBLE\_EQ(result.a, -71'299.5215);

EXPECT\_DOUBLE\_EQ(result.b, 3'680.22443);

EXPECT\_DOUBLE\_EQ(result.c, 58'201.17829);

EXPECT\_DOUBLE\_EQ(result.d, 384'977.1121);

}

TEST(Minus, HandlesZeroInput) {

CPolynom4 C1(pol\_3), C2(pol\_2), Diff;

Diff = C1 - C2;

SPolynom result = Diff.GetPolynom();

EXPECT\_DOUBLE\_EQ(result.a, -pol\_2.a);

EXPECT\_DOUBLE\_EQ(result.b, -pol\_2.b);

EXPECT\_DOUBLE\_EQ(result.c, -pol\_2.c);

EXPECT\_DOUBLE\_EQ(result.d, -pol\_2.d);

}

TEST(Minus, HandlesBigInput) {

CPolynom4 C1(pol\_5), C2(pol\_5), Diff;

Diff = C1 - C2;

SPolynom result = Diff.GetPolynom();

EXPECT\_DOUBLE\_EQ(result.a, 0);

EXPECT\_DOUBLE\_EQ(result.b, 0);

EXPECT\_DOUBLE\_EQ(result.c, 0);

EXPECT\_DOUBLE\_EQ(result.d, 0);

}

TEST(Minus, HandlesNegativeInput) {

CPolynom4 C1(pol\_4), C2(pol\_1), Diff;

Diff = C1 - C2;

SPolynom result = Diff.GetPolynom();

EXPECT\_DOUBLE\_EQ(result.a, -4'474.5876);

EXPECT\_DOUBLE\_EQ(result.b, -11'441.8926);

EXPECT\_DOUBLE\_EQ(result.c, -65'895.26537);

EXPECT\_DOUBLE\_EQ(result.d, -409'434.82842);

}

TEST(operator90, HandlesInput) {

CPolynom4 C1(pol\_4);

ldouble result = C1(365.422);

EXPECT\_DOUBLE\_EQ(result, -114'766'454'478.6443750189008);

}

TEST(Inverse, HandlesNormalInput) {

CPolynom4 C1(pol\_4);

CPolynom4 result = !C1;

SPolynom pol = result.GetPolynom();

EXPECT\_DOUBLE\_EQ(pol.a, 2346.2346);

EXPECT\_DOUBLE\_EQ(pol.b, 2087.5473);

EXPECT\_DOUBLE\_EQ(pol.c, 2326.5428);

EXPECT\_DOUBLE\_EQ(pol.d, 23873.45372);

}

TEST(Inverse, HandlesZero) {

CPolynom4 C1(pol\_3);

CPolynom4 result = !C1;

SPolynom pol = result.GetPolynom();

EXPECT\_DOUBLE\_EQ(pol.a, 0);

EXPECT\_DOUBLE\_EQ(pol.b, 0);

EXPECT\_DOUBLE\_EQ(pol.c, 0);

EXPECT\_DOUBLE\_EQ(pol.d, 0);

}

TEST(CPolynom4Test, AccessOperator) {

CPolynom4 p;

p[0] = 2128.353;

p[1] = 9354.3453;

p[2] = 63568.72257;

p[3] = 385561.3747;

ASSERT\_DOUBLE\_EQ(p[0], 2128.353);

ASSERT\_DOUBLE\_EQ(p[1], 9354.3453);

ASSERT\_DOUBLE\_EQ(p[2], 63568.72257);

ASSERT\_DOUBLE\_EQ(p[3], 385561.3747);

}

// Test the << operator

TEST(CPolynom4Test, OutputOperator) {

CPolynom4 p;

p[0] = 1.0;

p[1] = 2.0;

p[2] = 3.0;

p[3] = 4.0;

std::ostringstream oss;

oss << p;

EXPECT\_EQ(oss.str(), "\n\n(1)x^3 + (2)x^2 + (3)x + (4)\n\n");

}

// Test the >> operator

TEST(CPolynom4Test, InputOperator) {

CPolynom4 p;

std::istringstream iss("1.0 2.0 3.0 4.0");

iss >> p;

EXPECT\_DOUBLE\_EQ(p[0], 1.0);

EXPECT\_DOUBLE\_EQ(p[1], 2.0);

EXPECT\_DOUBLE\_EQ(p[2], 3.0);

EXPECT\_DOUBLE\_EQ(p[3], 4.0);

}

// Test the \* operator

TEST(CPolynom4Test, ScalarMultiplicationOperator) {

CPolynom4 p(pol\_4);

CPolynom4 result = p \* 3.0;

EXPECT\_DOUBLE\_EQ(result[0], -7'038.7038);

EXPECT\_DOUBLE\_EQ(result[1], -6'262.6419);

EXPECT\_DOUBLE\_EQ(result[2], -6'979.6284);

EXPECT\_DOUBLE\_EQ(result[3], -71'620.36116);

}

// Test the == operator

TEST(CPolynom4Test, EqualityOperator) {

CPolynom4 p1(pol\_1);

CPolynom4 p2;

p2[0] = 2128.353;

p2[1] = 9354.3453;

p2[2] = 63568.72257;

p2[3] = 385561.3747;

EXPECT\_TRUE(p1 == p2);

}

// Test the != operator

TEST(CPolynom4Test, InequalityOperator) {

CPolynom4 p1(pol\_1);

CPolynom4 p2;

p2[0] = 2128.353;

p2[1] = 9354.3453;

p2[2] = 63568.72257;

p2[3] = 385561.3748;

EXPECT\_TRUE(p1 != p2);

}

## Lib:

### RootsHeader.h:

#pragma once

#include <cmath>

using namespace std;

typedef long double ldouble;

typedef struct{

ldouble a;

ldouble b;

ldouble c;

ldouble d;

} SPolynom;

typedef struct {

ldouble roots[3];

int num;

} SResult;

class RootsFinder {

private:

ldouble leftLim, rightLim, eps, uLeft, uRight;

SResult result;

SPolynom polynom;

ldouble f(ldouble x);

int D(SPolynom pol);

ldouble FindSecant(ldouble leftLim, ldouble rightLim);

bool SetLimits(bool isFirst);

int numOfRoots;

public:

RootsFinder(SPolynom polynom);

RootsFinder(ldouble leftLim, ldouble rightLim, SPolynom polynom);

RootsFinder(ldouble leftLim, ldouble rightLim, ldouble epsilon, SPolynom polynom);

SResult FindBySecant();

SResult FindByDichotomy();

};

### GetRoots.cpp:

#include <iostream>

#include "RootsHeader.h"

ldouble RootsFinder::f(ldouble x) {

return polynom.a \* x \* x \* x + polynom.b \* x \* x + polynom.c \* x + polynom.d;

}

int RootsFinder::D(SPolynom pol) {

int result = 0;

ldouble D = 0;

if (fabs(pol.a) <= 0.000000000000001) {

if (fabs(pol.b) <= 0.000000000000001 && fabs(pol.c) <= 0.000000000000001) {

result = 0;

cout << "Equation has no roots" << endl << endl;

}

else if (fabs(pol.b) <= 0.000000000000001) {

result = 1;

}

else {

D = pol.c \* pol.c - 4 \* pol.b \* pol.d;

if (D < 0)

result = 0;

else if (fabs(D) < 0.000000000000001)

result = 1;

else

result = 2;

}

}

else if (fabs(pol.a) > 0.000000000000001 && fabs(pol.b) <= 0.000000000000001 && fabs(pol.c) <= 0.000000000000001 && fabs(pol.d) <= 0.000000000000001) {

result = 1;

}

else {

ldouble r = pol.b / pol.a, s = pol.c / pol.a, t = pol.d / pol.a;

ldouble p = (3 \* s - r \* r) / 3, q = (2 \* r \* r \* r) / 27 - r \* s / 3 + t;

D = p \* p \* p / 27 + q \* q / 4;

if (D > 0)

result = 1;

else if (fabs(D) < 0.000000000000001)

result = 2;

else

result = 3;

}

return result;

}

ldouble RootsFinder::FindSecant(ldouble leftLim, ldouble rightLim) {

return leftLim - (f(leftLim) \* (rightLim - leftLim)) / (f(rightLim) - f(leftLim));

}

SResult RootsFinder::FindBySecant() {

ldouble savedRightLim = rightLim;

bool resultIsFound = false, isFirst = true;

while (numOfRoots) {

resultIsFound = SetLimits(isFirst);

isFirst = false;

if (!result.num)

break;

if (resultIsFound) {

result.roots[numOfRoots - 1] = leftLim;

}

else {

result.roots[numOfRoots - 1] = FindSecant(leftLim, rightLim);

ldouble staticPoint = 0;

ldouble xPrev = result.roots[numOfRoots - 1];

while (!resultIsFound) {

if (f(result.roots[numOfRoots - 1]) \* f(leftLim) <= 0)

staticPoint = leftLim;

else

staticPoint = rightLim;

xPrev = result.roots[numOfRoots - 1];

result.roots[numOfRoots - 1] = FindSecant(result.roots[numOfRoots - 1], staticPoint);

if (fabs(result.roots[numOfRoots - 1] - xPrev) <= eps)

resultIsFound = true;

}

}

numOfRoots--;

resultIsFound = false;

leftLim = rightLim;

rightLim = savedRightLim;

}

return result;

}

SResult RootsFinder::FindByDichotomy()

{

ldouble midLim = 0, savedRightLim = rightLim;

bool BordersAreSetting = true, resultIsFound = false, isFirst = true;

while (numOfRoots) {

resultIsFound = SetLimits(isFirst);

isFirst = false;

if (!result.num)

break;

if (resultIsFound)

midLim = leftLim;

while (!resultIsFound) {

midLim = (leftLim + rightLim) / 2;

if (f(leftLim) \* f(midLim) <= 0) {

rightLim = midLim;

}

else {

leftLim = midLim;

}

midLim = (leftLim + rightLim) / 2;

if (fabs(leftLim - rightLim) < eps)

resultIsFound = true;

}

result.roots[numOfRoots - 1] = midLim;

numOfRoots--;

resultIsFound = false;

leftLim = rightLim;

rightLim = savedRightLim;

}

return result;

}

bool RootsFinder::SetLimits(bool isFirst) {

bool onGoing = true, resultIsFound = false;

ldouble step = (fabs(leftLim) + fabs(rightLim)) / 10, fX = 0, fLeft = 0;

if (leftLim > rightLim) {

ldouble temp = leftLim;

leftLim = rightLim;

rightLim = temp;

}

if (step >= 1)

step = 1;

fLeft = f(leftLim);

if (fLeft == 0 && isFirst) {

resultIsFound = true;

onGoing = false;

}

while (onGoing) {

for (ldouble x = leftLim + step; x < rightLim; x += step) {

fX = f(x);

if (fabs(fX) <= eps) {

leftLim = x;

ldouble k = x + eps;

while (fabs(k) <= eps)

k += eps;

rightLim = k;

resultIsFound = true;

onGoing = false;

break;

}

if (fLeft \* fX <= 0) {

leftLim = x - step;

rightLim = x;

onGoing = false;

break;

}

fLeft = fX;

}

step /= 10;

if (step <= eps) {

result.num = 0;

result.roots[0] = NULL;

result.roots[1] = NULL;

result.roots[2] = NULL;

cout << "There are no roots in range [" << uLeft << "; " << uRight << "], or not all of them. Please, set your range" << endl << endl;

onGoing = false;

}

}

return resultIsFound;

}

RootsFinder::RootsFinder(SPolynom polynom) {

this->leftLim = -100;

this->rightLim = 100;

this->polynom = polynom;

eps = 0.000001;

uLeft = leftLim;

uRight = rightLim;

result.roots[0] = NAN;

result.roots[1] = NAN;

result.roots[2] = NAN;

result.num = D(polynom);

numOfRoots = result.num;

}

RootsFinder::RootsFinder(ldouble leftLim, ldouble rightLim, SPolynom polynom) {

this->leftLim = leftLim;

this->rightLim = rightLim;

this->polynom = polynom;

eps = 0.000001;

uLeft = leftLim;

uRight = rightLim;

result.roots[0] = NAN;

result.roots[1] = NAN;

result.roots[2] = NAN;

result.num = D(polynom);

numOfRoots = result.num;

}

RootsFinder::RootsFinder(ldouble leftLim, ldouble rightLim, ldouble epsilon, SPolynom polynom) {

this->leftLim = leftLim;

this->rightLim = rightLim;

this->polynom = polynom;

eps = epsilon;

uLeft = leftLim;

uRight = rightLim;

result.roots[0] = NAN;

result.roots[1] = NAN;

result.roots[2] = NAN;

result.num = D(polynom);

numOfRoots = result.num;

}

# Результат виконання

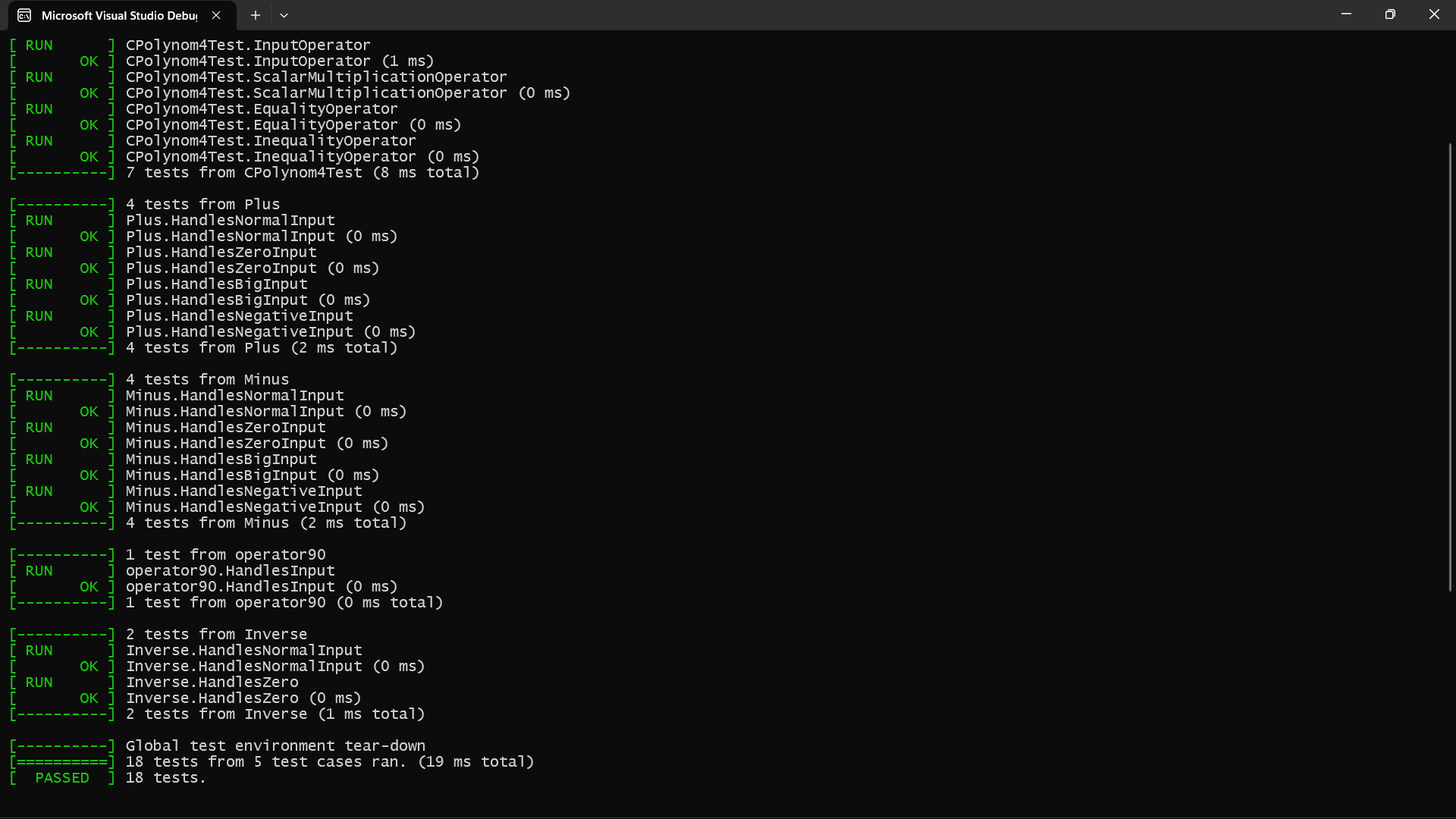


Рис 1. Виконання тестів

# Висновок

Під час виконання лабораторної роботи я навчився використовувати механізм перевантаження функцій та операторів, створювати та використовувати дружні функції, ознайомився зі статичними полями і методами та навчився їх використовувати.

Зміст

[Теоретичні відомості 2](#_Toc130162182)

[Завдання 2](#_Toc130162183)

[Код програми 2](#_Toc130162184)

[Lab\_03\_OOP: 2](#_Toc130162185)

[UI: 2](#_Toc130162186)

[Lab\_03\_OOP.cpp: 2](#_Toc130162187)

[PolynomHeader.h: 2](#_Toc130162188)

[Lab\_03\_Tests\_OOP: 2](#_Toc130162189)

[pch.h: 2](#_Toc130162190)

[test.cpp: 2](#_Toc130162191)

[Lib: 2](#_Toc130162192)

[RootsHeader.h: 2](#_Toc130162193)

[GetRoots.cpp: 2](#_Toc130162194)

[Результат виконання 2](#_Toc130162195)

[Висновок 2](#_Toc130162196)

[Зміст 2](#_Toc130162197)