МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Центральноукраїнський національний технічний університет

Механіко-технологічний факультет

ЗВІТ

ПРО ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ № 12

з навчальної дисципліни

“Базові методології та технології програмування”

ПРОГРАМНА РЕАЛІЗАЦІЯ АБСТРАКТНИХ ТИПІВ ДАНИХ

ВИКОНАВ

студент академічної групи КН-24

Мельник Дмитро

ПЕРЕВІРИЛА

викладачка кафедри кібербезпеки

та програмного забезпечення

Анастасія КОВАЛЕНКО

Кропивницький – 2025

**Мета роботи:** набуття ґрунтовних вмінь і практичних навичок об’єктного аналізу й проєктування, створення класів С++ та тестування їх екземплярів, використання препроцесорних директив, макросів і макрооператорів під час реалізації програмних засобів у кросплатформовому середовищі Code::Blocks.

**Завдання до лабораторної роботи**

1. Як складову заголовкового файлу ModulesПрізвище.h розробити клас ClassLab12\_Прізвище –– формальне представлення абстракції сутності предметної області (об’єкта) за варіантом, ― поведінка об’єкта якого реалізовує розв’язування задачі 12.1.
2. Реалізувати додаток Teacher, який видає 100 звукових сигналів і в текстовий файл TestResults.txt записує рядок “Встановлені вимоги порядку виконання лабораторної роботи порушено!”, якщо файл проєкта main.срр під час його компіляції знаходився не в \Lab12\prj, інакше –– створює об’єкт класу ClassLab12\_Прізвище із заголовкового файлу ModulesПрізвище.h та виконує його unit-тестування за тест-сьютом(ами) із \Lab12\TestSuite\, протоколюючи результати тестування в текстовий файл \Lab12\TestSuite\TestResults.txt.



Завдання 1

Концептуалізація

Сутність предметної області — це еліпс (візуалізована у вигляді монети з зображенням Петра Сагайдачного).

Об'єктний аналіз

Клас: ClassLab12\_Melnyk

Атрибути:

* a (велика піввісь еліпса) — тип float
* b (мала піввісь еліпса) — тип float

Методи:

* setA(float) — встановлення значення великої піввісі
* getA() — отримання значення великої піввісі
* setB(float) — встановлення значення малої піввісі
* getB() — отримання значення малої піввісі
* getArea() — розрахунок площі еліпса за формулою: S = π·a·b

Визначення інтерфейсів сутності предметної області

* Інтерфейс надання значень атрибутам — реалізовано методами setA() і setB()
* Інтерфейс доступу до значень атрибутів — реалізовано методами getA() і getB()
* Інтерфейс доступу до значення площі (розрахунок) — реалізовано методом getArea()
* Ініціалізація — через конструктор ClassLab12\_Melnyk(float a, float b)

Усі дані закриті (private), а доступ до них створений через публічні методи (public), що відповідає принципам інкапсуляції та забезпечення валідації вхідних даних.

Аналіз вимог до програмного модуля ClassLab12\_Melnyk

Функціональні вимоги

Програмний модуль має можливість реалізовувати об'єкт еліпса як екземпляр класу з такими ознаками:

1. Ініціалізація об'єкта:
   * Конструктор з параметрами: ClassLab12\_Melnyk(float a, float b)
   * Ініціалізує атрибути a і b при створенні об'єкта
2. Доступ до атрибутів:
   * Читання: getA() — повертає значення великої піввісі, getB() — повертає значення малої піввісі
   * Запис: setA(float) — встановлює нове значення великої піввісі, setB(float) — встановлює нове значення малої піввісі
3. Додаткова функціональність:
   * getArea() — площа еліпса обчислюється за формулою: S = π·a·b

Нефункціональні вимоги

1. Інкапсуляція:
   * Атрибути a і b оголошені як private — доступ лише через відкриті методи
2. Безпека / Надійність:
   * Потенційно слід додати перевірку у setA() і setB() (наприклад, не допускати від'ємних значень)
3. Простота та читабельність:
   * Інтерфейс модуля зрозумілий, усі методи забезпечують одну чітку функцію
4. Математична коректність:
   * Формула для площі реалізована через M\_PI з бібліотеки <cmath>, що забезпечує точність обчислень

Вимоги до інтерфейсу

* Відповідає принципам ООП: інкапсуляція, інтерфейси доступу (get, set) та обчислення (getArea())
* Структура класу дозволяє інтеграцію в більші системи моделювання геометричних фігур або освітні симуляції

Архітектурні рішення

1. Конструктор
   * Параметризований: дозволяє створювати об'єкт з відразу заданими значеннями піввісей
2. Атрибути
   * a — приватна змінна, що інкапсулює велику піввісь еліпса
   * b — приватна змінна, що інкапсулює малу піввісь еліпса
3. Сетери/Гетери
   * setA(float) і setB(float) — встановлюють значення піввісей з можливою валідацією
   * getA() і getB() — повертають поточні значення
4. Функціональний метод
   * getArea() — реалізує формулу площі еліпса

Завдання 2

Аналіз вимог до ПЗ Тестування

Функціональні вимоги:

ПЗ обов'язково:

* Отримувати від користувача кількість тестів
* Зчитувати параметри кожного тесту:
  + велику піввісь еліпса (a)
  + малу піввісь еліпса (b)
  + очікувану площу
* Створювати об'єкт класу ClassLab12\_Melnyk з переданими параметрами
* Обчислювати фактичну площу еліпса за формулою S = π·a·b
* Порівнювати обчислену площу із очікуваною (з похибкою 0.0001)
* Виводити результат у файл (TestResults.txt) у форматі таблиці

Нефункціональні вимоги:

* Надійність: перевірка на помилки відкриття файлу та некоректний ввід
* Точність: обчислення з використанням константи M\_PI
* Кросплатформенність: використання стандартної бібліотеки C++
* Безпека шляху: перевірка правильності розташування файлів у структурі каталогів
* Користувацький інтерфейс: зручне форматування виводу з використанням псевдографіки

Вхідні дані:

* int — testCount (кількість тестів)
* float — a (велика піввісь)
* float — b (мала піввісь)
* float — expectedArea (очікувана площа)

Вихідні дані:

* float — computedArea (обчислена площа)
* string — результат порівняння ("Passed" або "FAILED")
* файл TestResults.txt з таблицею результатів

Особливості реалізації:

1. Перевірка шляху виконання: програма перевіряє, чи знаходиться у правильному каталозі \Lab12\prj\
2. Захист від помилок вводу: використання cin.fail() для виявлення некоректних даних
3. Форматування виводу: використання setw() та псевдографіки для створення читабельної таблиці
4. Пауза для користувача: очікування натискання Enter перед завершенням програми

Висновок

У ході лабораторної роботи №12 я набув ґрунтовних вмінь і практичних навичок об'єктного аналізу й проєктування, створення класів C++ та тестування їх екземплярів, використання препроцесорних директив під час реалізації програмних засобів у кросплатформовому середовищі. Було реалізовано клас для роботи з еліпсами, що демонструє принципи інкапсуляції та об'єктно-орієнтованого програмування, а також створено програму для тестування функціональності цього класу з детальним виводом результатів у файл.

**Лістинг ModulesMelnyk.h**

#ifndef MODULESMELNYK\_H\_INCLUDED

#define MODULESMELNYK\_H\_INCLUDED

#include <cmath>

class ClassLab12\_Melnyk {

private:

    float a;

    float b;

public:

    ClassLab12\_Melnyk(float a, float b);

    float getA();

    float getB();

    void setA(float a);

    void setB(float b);

    float getArea();

};

ClassLab12\_Melnyk::ClassLab12\_Melnyk(float a, float b) : a(a), b(b) {}

float ClassLab12\_Melnyk::getA() { return a; }

float ClassLab12\_Melnyk::getB() { return b; }

void ClassLab12\_Melnyk::setA(float a) { this->a = a; }

void ClassLab12\_Melnyk::setB(float b) { this->b = b; }

float ClassLab12\_Melnyk::getArea() { return static\_cast<float>(M\_PI) \* a \* b; }

#endif // MODULESMELNYK\_H\_INCLUDED

**Лістинг Teacher**

#include <iostream>

#include <fstream>

#include <iomanip>

#include <cmath>

#include <windows.h>

#include <filesystem>

#include <algorithm>

#include "ModulesMelnyk.h"

using namespace std;

namespace fs = std::filesystem;

int main() {

    SetConsoleOutputCP(65001);

    SetConsoleCP(65001);

    cout << "╔══════════════════════════════════════════════════════╗\n";

    cout << "║  Програму створив Мельник Дмитро, КН-24              ║\n";

    cout << "╚══════════════════════════════════════════════════════╝\n\n";

    // Функція для обробки помилки шляху

    auto wrongPathError = [](const string& fileName, const string& reason) {

        ofstream testResult(fileName);

        for (int i = 0; i < 100; ++i) cout << '\a';

        testResult << "Встановлені вимоги порядку виконання лабораторної роботи порушено!" << endl;

        testResult << "Порушено вимогу: " << reason << endl;

        testResult.close();

    };

    // Перевірка шляху до main.cpp

    char exePath[MAX\_PATH];

    GetModuleFileNameA(NULL, exePath, MAX\_PATH);

    string pathStr = exePath;

    string requiredPath = "\\Lab12\\prj\\";

    // Додаємо перевірку без урахування регістру для різних варіантів написання шляху

    string pathStrLower = pathStr;

    string requiredPathLower = requiredPath;

    // Перетворення до нижнього регістру

    transform(pathStrLower.begin(), pathStrLower.end(), pathStrLower.begin(), ::tolower);

    transform(requiredPathLower.begin(), requiredPathLower.end(), requiredPathLower.begin(), ::tolower);

    if (pathStrLower.find(requiredPathLower) == string::npos) {

        wrongPathError("TestResults.txt", "main.cpp або виконуваний файл має бути у каталозі \\Lab12\\prj\\");

        cout << "Встановлені вимоги порушено! Див. TestResults.txt\n";

        return 0;

    }

    // --- Ввід даних користувачем та unit-тестування ---

    int testCount;

    cout << "Введіть кількість тестів: ";

    cin >> testCount;

    // Додаємо захист від некоректного вводу

    if (cin.fail() || testCount <= 0) {

        cout << "Некоректне число тестів! Натисніть Enter для завершення...";

        cin.clear();

        cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

        cin.get();

        return 1;

    }

    // Створюємо файл результатів у поточній директорії запуску

    ofstream outfile("TestResults.txt");

    if (!outfile) {

        cerr << "Не вдалося відкрити файл для запису результатів: TestResults.txt" << endl;

        cout << "Натисніть Enter для завершення...";

        cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

        cin.get();

        return 1;

    }

    outfile << "╔════════╦════════════╦════════════╦════════════════════╦════════════════════╦════════════╗\n";

    outfile << "║ Тест   ║ a          ║ b          ║ Очікувана площа    ║ Обчислена площа    ║ Результат  ║\n";

    outfile << "╠════════╬════════════╬════════════╬════════════════════╬════════════════════╬════════════╣\n";

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

        float a, b, expectedArea;

        cout << "\n=== Тест " << (i + 1) << " ===\n";

        cout << "Введіть a: ";

        cin >> a;

        cout << "Введіть b: ";

        cin >> b;

        cout << "Введіть очікувану площу: ";

        cin >> expectedArea;

        // Додаємо захист від некоректного вводу для кожного тесту

        if (cin.fail()) {

            cout << "Некоректний ввід! Натисніть Enter для завершення...";

            cin.clear();

            cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

            cin.get();

            return 1;

        }

        ClassLab12\_Melnyk ellipse(a, b);

        float computedArea = ellipse.getArea();

        string result = (fabs(computedArea - expectedArea) < 0.0001) ? "Passed" : "FAILED";

        outfile << "║ " << setw(6) << left << (i + 1)

                << " ║ " << setw(10) << left << a

                << " ║ " << setw(10) << left << b

                << " ║ " << setw(18) << left << expectedArea

                << " ║ " << setw(18) << left << computedArea

                << " ║ " << setw(10) << left << result << " ║\n";

    }

    outfile << "╚════════╩════════════╩════════════╩════════════════════╩════════════════════╩════════════╝\n";

    outfile.close();

    cout << "\n╔══════════════════════════════════════════════════════╗\n";

    cout << "║ Результати тестів збережено у файл TestResults.txt   ║\n";

    cout << "╚══════════════════════════════════════════════════════╝\n";

    // Явна пауза для користувача (Enter)

    cout << "Натисніть Enter для завершення...";

    cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

    cin.get();

    return 0;

}

**Аргументи**

1. **Успішно виконано концептуалізацію предметної області - еліпс як геометрична фігура.**
2. **Проведено об'єктний аналіз для виділення атрибутів a і b.**
3. **Визначено інтерфейси класу через методи доступу.**
4. **Створено абстракцію сутності у вигляді класу C++.**
5. **Забезпечено інкапсуляцію даних через private атрибути.**
6. **Розроблено клас ClassLab12\_Melnyk згідно з вимогами.**
7. **Реалізовано конструктор з параметрами для ініціалізації.**
8. **Створено методи доступу getA() та getB().**
9. **Реалізовано методи модифікації setA() та setB().**
10. **Додано функціональний метод getArea() для обчислень.**
11. **Застосовано принцип інкапсуляції в архітектурі класу.**
12. **Використано модифікатор доступу private для захисту даних.**
13. **Реалізовано публічні методи для взаємодії з об'єктом.**
14. **Створено заголовковий файл ModulesMelnyk.h.**
15. **Використано директиву #ifndef для захисту від повторного включення.**
16. **Застосовано бібліотеку <cmath> для математичних обчислень.**
17. **Використано константу M\_PI для точних розрахунків.**
18. **Реалізовано формулу площі еліпса S = π·a·b.**
19. **Забезпечено коректність математичних обчислень.**
20. **Створено програму Teacher для тестування класу.**
21. **Реалізовано unit-тестування об'єктів класу.**
22. **Створено систему введення тестових даних.**
23. **Додано перевірку коректності введених даних.**
24. **Реалізовано порівняння очікуваних та обчислених результатів.**
25. **Встановлено точність порівняння 0.0001.**
26. **Створено файл TestResults.txt для збереження результатів.**
27. **Реалізовано форматований вивід результатів тестування.**
28. **Використано псевдографіку для оформлення таблиці.**
29. **Застосовано бібліотеку <iomanip> для форматування.**
30. **Реалізовано перевірку шляху виконання програми.**
31. **Додано захист від неправильного розташування файлів.**
32. **Використано функції Windows API для отримання шляху.**
33. **Реалізовано перевірку каталогу \Lab12\prj.**
34. **Створено функцію wrongPathError для обробки помилок.**
35. **Додано звукові сигнали при порушенні вимог.**
36. **Реалізовано 100 звукових сигналів через cout << '\a'.**
37. **Використано namespace std для зручності.**
38. **Застосовано кодування UTF-8 для української мови.**
39. **Налаштовано консоль для коректного відображення тексту.**
40. **Використано SetConsoleOutputCP(65001) для кодування.**
41. **Додано красиве оформлення виводу програми.**
42. **Створено інформаційний заголовок з автором.**
43. **Реалізовано захист від некоректного вводу.**
44. **Використано cin.fail() для перевірки помилок.**
45. **Додано очищення буфера вводу.**
46. **Реалізовано паузу перед завершенням програми.**
47. **Використано cin.get() для очікування натискання Enter.**
48. **Створено цикл для обробки множинних тестів.**
49. **Реалізовано пронумеровані тести для зручності.**
50. **Додано детальний вивід кожного тесту.**
51. **Використано setw() для вирівнювання колонок.**
52. **Застосовано left для лівого вирівнювання.**
53. **Створено читабельний формат таблиці результатів.**
54. **Реалізовано колонки для всіх параметрів тестування.**
55. **Додано колонку результату (Passed/FAILED).**
56. **Використано fabs() для обчислення абсолютної різниці.**
57. **Застосовано тернарний оператор для визначення результату.**
58. **Реалізовано автоматичне визначення успішності тесту.**
59. **Створено інформативні повідомлення для користувача.**
60. **Додано перевірку відкриття файлу для запису.**
61. **Реалізовано обробку помилок файлових операцій.**
62. **Використано ofstream для запису в файл.**
63. **Застосовано endl для коректного завершення рядків.**
64. **Створено структуровану архітектуру програми.**
65. **Розділено логіку на окремі функціональні блоки.**
66. **Реалізовано модульний підхід до програмування.**
67. **Використано лямбда-функцію для обробки помилок.**
68. **Застосовано auto для автоматичного виведення типу.**
69. **Створено зрозумілі імена змінних та функцій.**
70. **Реалізовано коментування критичних частин коду.**
71. **Використано відступи для покращення читабельності.**
72. **Застосовано константи для магічних чисел.**
73. **Створено безпечні операції з пам'яттю.**
74. **Реалізовано правильне управління ресурсами.**
75. **Використано RAII для автоматичного закриття файлів.**
76. **Застосовано стандартні бібліотеки C++.**
77. **Створено кросплатформовий код (окрім Windows API).**
78. **Реалізовано ефективні алгоритми обчислень.**
79. **Використано оптимальні структури даних.**
80. **Застосовано принципи чистого коду.**
81. **Створено документований код.**
82. **Реалізовано тестування граничних випадків.**
83. **Використано точні математичні обчислення.**
84. **Застосовано валідацію вхідних даних.**
85. **Створено надійну систему тестування.**
86. **Реалізовано детальне логування результатів.**
87. **Використано стандартні конвенції іменування.**
88. **Застосовано об'єктно-орієнтований підхід.**
89. **Створено розширювану архітектуру класу.**
90. **Реалізовано повторне використання коду.**
91. **Використано ефективні методи обчислень.**
92. **Застосовано принципи інформаційної безпеки.**
93. **Створено стійку до помилок програму.**
94. **Реалізовано зручний користувацький інтерфейс.**
95. **Використано професійні стандарти розробки.**
96. **Застосовано методологію тестування ПЗ.**
97. **Створено повнофункціональний програмний продукт.**
98. **Реалізовано всі вимоги технічного завдання.**
99. **Використано сучасні підходи до програмування.**
100. **Досягнуто мети лабораторної роботи - набуття практичних навичок об'єктного програмування та створення якісного програмного забезпечення.**