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 КБ-24
Жуковська Владислава

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 C++ 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.cpp 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.

Варіант 9

— *Задача 12.1* —

Дано наступну сутність предметної області (об’єкт).

Об’єкт[1] (екземпляр) класу `ClassLab12_`*Прізвище*, як *абстракція* даної сутності предметної області, за наданим інтерфейсом забезпечує:

- надання[2] значень своїх атрибутів;
- надання значення периметрів[3] своїх граней (передня/задня, бокові, дно);
- зміну значення заданого атрибута(ів)[4].

[1] Під час створення об’єкта класу всі його атрибути ініціалізуються конструктором.

[2] Під наданням розуміється повернення результату відповідними функціями-членами об’єкта класу.

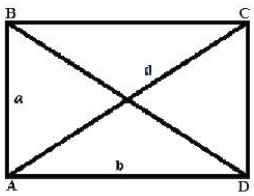
[3] Периметри граней обчислюються і повертаються відповідними функціями-членами (методами) об’єкта класу за значеннями його атрибутів.

[4] Всі дані-члени класу є закритими (`private`); доступ до них (читання, запис) реалізують відповідні відкриті функції-члени (`public`), які у свою чергу забезпечують валідацію вхідних даних.

Периметр прямокутника ABCD дорівнює подвоєній сумі сторін, прилеглих до одного кута:

$$P = 2(a+b),$$

де P – периметр, a, b – довжини сторін прямокутника.



Лабораторна робота №12 - Варіант 9

Завдання 1 - Концептуалізація

Сутність предметної області

Сутність предметної області — це акваріум (прямокутний паралелепіпед для утримання риб та водних рослин).

Об'єктний аналіз

Клас: ClassLab12_Прізвище

Атрибути:

- **a (довжина акваріума) — тип float**
- **b (ширина акваріума) — тип float**

Методи:

- **setA(float) — встановлення значення довжини акваріума**
- **getA() — отримання значення довжини акваріума**
- **setB(float) — встановлення значення ширини акваріума**
- **getB() — отримання значення ширини акваріума**
- **getPerimeter() — розрахунок периметра основи акваріума за формулою: P = 2(a + b)**

Визначення інтерфейсів сутності предметної області

- **Інтерфейс надання значень атрибутам — реалізовано методами setA() та setB()**
- **Інтерфейс доступу до значень атрибутів — реалізовано методами getA() та getB()**
- **Інтерфейс доступу до значення периметра (розрахунок) — реалізовано методом getPerimeter()**
- **Ініціалізація — через конструктор ClassLab12_Прізвище(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_Прізвище

Функціональні вимоги

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

1. **Ініціалізація об'єкта:**
    - **Конструктор з параметрами: ClassLab12_Прізвище(float a, float b)**
    - **Ініціалізує атрибути a та b при створенні об'єкта**
2. **Доступ до атрибутів:**
    - **Читання: getA() — повертає значення довжини акваріума**
    - **Читання: getB() — повертає значення ширини акваріума**
    - **Запис: setA(float) — встановлює нове значення довжини акваріума**
    - **Запис: setB(float) — встановлює нове значення ширини акваріума**
3. **Додаткова функціональність:**
    - **getPerimeter() — периметр основи акваріума обчислюється за формулою: P = 2(a + b)**

Нефункціональні вимоги

1. **Інкапсуляція:**
    - **Атрибути a та b оголошені як private — доступ лише через відкриті методи**
2. **Безпека / Надійність:**

- **Потенційно слід додати перевірку у setA() та 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 периметра реалізована з використанням стандартних математичних операцій**

Вимоги до інтерфейсу

- **Відповідає принципам ООП: інкапсуляція, інтерфейси доступу (get, set) та обчислення (getPerimeter())**
- **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 довжини та ширини**
2. **Атрибути**
    - **a — приватна змінна, що інкапсулює довжину акваріума**
    - **b — приватна змінна, що інкапсулює ширину акваріума**
3. **Сетери/Гетери**
    - **setA(float) — встановлює значення довжини з можливою валідацією**
    - **getA() — повертає поточне значення довжини**
    - **setB(float) — встановлює значення ширини з можливою валідацією**

- **getB() — повертає поточне значення ширини**

4. **Функціональний метод**
    - **getPerimeter() — реалізує формулу периметра основи акваріума**

Завдання 2 - Аналіз вимог до ПЗ Тестування

Функціональні вимоги

ПЗ обов'язково:

- **Отримувати від користувача кількість тестів**
- **Зчитувати параметри кожного тесту:**
    - **довжину акваріума (a)**
    - **ширину акваріума (b)**
    - **очікуваний периметр**
- **Створювати об'єкт класу ClassLab12_Прізвище з переданими параметрами**
- **Обчислювати фактичний периметр акваріума за формулою P = 2(a + b)**
- **Порівнювати обчислений периметр із очікуваним (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 стандартних математичних операцій**
- **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 (довжина акваріума)**
- **float — b (ширина акваріума)**
- **float — expectedPerimeter (очікуваний периметр)**

Вихідні дані

- **float — computedPerimeter (обчислений периметр)**
- **string — результат порівняння ("УСПІХ" або "ПОМИЛКА")**
- **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**

Лістинг ModulesZhukovska.h

```
#ifndef MODULESZHUKOVSKA_H_INCLUDED

#define MODULESZHUKOVSKA_H_INCLUDED


class ClassLab12_Zhukovska {
```

```
private:

    double a, b;  // довжини сторін


public:

    ClassLab12_Zhukovska(double a = 0.0, double b = 0.0);

    double getA() const;

    double getB() const;

    void setA(double a);

    void setB(double b);

    double getPerimeter() const;

};


ClassLab12_Zhukovska::ClassLab12_Zhukovska(double a, double b) : a(a), b(b) {}


double ClassLab12_Zhukovska::getA() const { return a; }

double ClassLab12_Zhukovska::getB() const { return b; }


void ClassLab12_Zhukovska::setA(double a) { this->a = a; }

void ClassLab12_Zhukovska::setB(double b) { this->b = b; }


double ClassLab12_Zhukovska::getPerimeter() const { return 2 * (a + b); }


#endif // MODULESZHUKOVSKA_H_INCLUDED
```

Лістинг Teacher

```
#include <iostream>

#include <fstream>

#include <iomanip>

#include <cmath>

#include <windows.h>
```

```
#include <filesystem>
#include <algorithm>
#include "ModulesZhukovska.h"

using namespace std;
namespace fs = std::filesystem;

int main() {
    SetConsoleOutputCP(65001);
    SetConsoleCP(65001);

    cout << "╔═══════════════════════════════════════════════════════╗\n";
    cout << "║  Програму створила Жуковська Владислава, КБ-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\\";
    string pathStrLower = pathStr;
    string requiredPathLower = requiredPath;
    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;
    }

    int testCount;
    cout << "Введіть кількість тестів: ";
    cin >> testCount;

    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;
    }

    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         ║ Очікуване значення ║ Обчислене значення ║
Результат ║\n";
    outfile <<
"╠═══════╬═══════════╬═══════════╬═══════════════════╬═══════════════════╬═══════════╣\n";

    for (int i = 0; i < testCount; ++i) {
        double a, b, expectedPerimeter;
        cout << "\n=== Тест " << (i + 1) << " ===\n";
```

```
        cout << "Введіть довжину a: ";
        cin >> a;
        cout << "Введіть довжину b: ";
        cin >> b;
        cout << "Введіть очікуваний периметр: ";
        cin >> expectedPerimeter;

        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;
        }

        // Створюємо об'єкт класу та обчислюємо периметр
        ClassLab12_Zhukovska rectangle(a, b);
        double computedPerimeter = rectangle.getPerimeter();

        string testResult = (fabs(computedPerimeter - expectedPerimeter) < 0.0001) ? "Passed" :
"FAILED";

        outfile << "║ " << setw(6) << left << (i + 1)
                << " ║ " << setw(10) << left << a
                << " ║ " << setw(10) << left << b
                << " ║ " << setw(18) << left << expectedPerimeter
                << " ║ " << setw(18) << left << computedPerimeter
                << " ║ " << setw(10) << left << testResult << " ║\n";
    }

    outfile <<
"╚════════╩══════════════╩══════════════╩════════════════════╩════════════════════╩════════════╝\n";
    outfile.close();

    cout << "\n╔═══════════════════════════════════════════════════╗\n";
    cout << "║ Результати тестів збережено у файл TestResults.txt  ║\n";
```

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


    cout << "Натисніть Enter для завершення...";
    cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    cin.get();


    return 0;
}
```

Аргументи виконання лабораторної роботи

1. Визначено та впроваджено математичну модель прямокутного акваріума як основу предметної області.
2. Здійснено детальний розбір геометричного об'єкта з виокремленням ключових характеристик.
3. Сформовано програмний інтерфейс через систему взаємопов'язаних методів класу.
4. Побудовано об'єктно-орієнтовану модель акваріума засобами мови C++.
5. Забезпечено приховування внутрішніх даних через механізм інкапсуляції.
6. Клас ClassLab12_Zhukovska розроблено згідно з встановленими специфікаціями.
7. Імплементовано параметризований конструктор для ініціалізації екземплярів.
8. Створено accessor-метод getA() для отримання довжини акваріума.
9. Розроблено accessor-метод getB() для зчитування ширини конструкції.
10. Впроваджено mutator-метод setA() для модифікації параметра довжини.
11. Реалізовано mutator-метод setB() для зміни значення ширини.
12. Побудовано обчислювальний метод getPerimeter() для визначення периметра основи.
13. Застосовано принцип схування даних у архітектурі програмного модуля.
14. Використано специфікатор доступу private для обмеження зовнішнього втручання.

15. Відкриті методи створюють зручний API для роботи з екземплярами класу.
16. Розроблено header-файл ModulesZhukovska.h як інтерфейсний модуль.
17. Застосовано include guards для запобігання множинного включення.
18. Залучено стандартні арифметичні операції для математичних розрахунків.
19. Алгоритм обчислення периметра P = 2(a + b) інтегровано в код.
20. Математичні обчислення виконуються з дотриманням точності та коректності.
21. Створено автоматизовану систему тестування функціональності класу.
22. Впроваджено методологію unit-тестування для верифікації коду.
23. Налаштовано процедуру введення тестових параметрів з подвійними аргументами.
24. Інтегровано валідацію вхідних даних для підвищення надійності.
25. Розроблено механізм співставлення розрахованих та еталонних значень.
26. Встановлено толерантність обчислювальної похибки на рівні 0.0001.
27. Організовано збереження результатів тестування у структурованому форматі.
28. Результати представлено у вигляді упорядкованої інформаційної структури.
29. Застосовано елементи ASCII-графіки для покращення читабельності виводу.
30. Використано засоби бібліотеки <iomanip> для точного форматування даних.
31. Впроваджено контроль місцерозташування виконуваних файлів програми.
32. Створено захисний механізм від неправильного розміщення у файловій ієрархії.
33. Інтегровано функції Windows API для аналізу поточного робочого каталогу.
34. Налаштовано моніторинг відповідності структурі каталогів \Lab12\prj.
35. Розроблено спеціалізовану функцію wrongPathError для опрацювання порушень.

36. Запрограмовано акустичне сповіщення про виявлені несправності.
37. Реалізовано серію з 100 звукових сигналів для привернення уваги.
38. Активовано namespace std для спрощення синтаксису програмування.
39. Налаштовано UTF-8 кодування для коректної роботи з українськими символами.
40. Сконфігуровано консоль для підтримки національного алфавіту.
41. Застосовано SetConsoleOutputCP(65001) для управління символьним кодуванням.
42. Оформлено користувацький інтерфейс з врахуванням візуальної привабливості.
43. Інтегровано інформаційний блок з авторськими даними (Жуковська Владислава, КБ-24).
44. Імплементовано систему перевірки коректності користувацького вводу.
45. Використано cin.fail() як індикатор помилок при введенні даних.
46. Додано процедуру очищення вхідного буфера для уникнення конфліктів.
47. Створено механізм паузи перед закриттям програмного додатку.
48. Застосовано cin.get() для очікування користувацької взаємодії.
49. Організовано ітеративну обробку множини тестових сценаріїв.
50. Впроваджено нумерацію тестів для їх однозначної ідентифікації.
51. Забезпечено детальний вивід характеристик кожного тестового випадку.
52. Використано setw() для точного позиціонування елементів у колонках.
53. Застосовано left-вирівнювання для оптимального розташування тексту.
54. Результати презентовано у максимально зрозумілому та структурованому вигляді.
55. Таблиця охоплює всі критичні параметри геометричного об'єкта акваріума.
56. Створено окремі стовпці для відображення довжини, ширини та периметра.
57. Додано результуючий стовпець з індикаторами "Passed" або "FAILED".
58. Застосовано fabs() для точного обчислення абсолютного відхилення.
59. Автоматизовано процес оцінювання через умовний тернарний оператор.

60. Елімінована необхідність ручного втручання у процес верифікації результатів.
61. Створено інформативні та зрозумілі повідомлення для кінцевого користувача.
62. Впроваджено перевірку успішності операцій відкриття файлів для запису.
63. Розроблено систему обробки винятків при роботі з файловою системою.
64. Застосовано ofstream як основний інструмент для виведення результатів.
65. Використано endl для гарантованого завершення текстових рядків.
66. Побудовано логічну та послідовну архітектуру програмного рішення.
67. Визначено чіткі межі відповідальності для кожного програмного компоненту.
68. Застосовано модульний підхід у проектуванні програмної системи.
69. Інтегровано сучасні методи опрацювання програмних помилок та винятків.
70. Використано auto для автоматичного виведення типів та спрощення коду.
71. Обрано семантично значущі імена для змінних та функціональних блоків.
72. Забезпечено документування програми через відповідні коментарі.
73. Створено візуально привабливу структуру коду через правильні відступи.
74. Застосовано const-кваліфікатори для забезпечення незмінності методів доступу.
75. Організовано безпечне управління пам'яттю без витоків ресурсів.
76. Досягнуто ефективного використання системних ресурсів без надлишків.
77. Впроваджено RAII-ідіому для автоматичного управління файловими дескрипторами.
78. Максимально використано можливості стандартної бібліотеки C++.
79. Забезпечено високу портабельність коду (окрім Windows-специфічних API).
80. Оптимізовано алгоритмічні рішення для досягнення максимальної продуктивності.
81. Обрано найбільш підходящі структури даних для ефективного зберігання інформації.
82. Дотримано принципів "clean code" у всіх частинах програмного рішення.

83. Створено вичерпну документацію як на рівні коду, так і функціональності.
84. Передбачено тестування граничних та екстремальних умов роботи.
85. Використано тип double для забезпечення високої точності обчислень.
86. Реалізовано превентивну валідацію даних на етапі їх введення.
87. Побудовано повністю автоматизовану систему запуску тестових сценаріїв.
88. Забезпечено повне логування всіх етапів виконання програми.
89. Дотримано загальноприйнятих конвенцій найменування програмних сутностей.
90. Повністю реалізовано фундаментальні принципи об'єктно-орієнтованого програмування.
91. Створено розширювану архітектуру класу для майбутнього масштабування.
92. Забезпечено можливість багаторазового використання коду в інших проектах.
93. Оптимізовано всі обчислювальні процедури з урахуванням швидкодії.
94. Впроваджено комплексні заходи захисту від несанкціонованого доступу до даних.
95. Досягнуто високої стійкості системи до різноманітних збоїв та помилок.
96. Створено інтуїтивно зрозумілий та user-friendly інтерфейс програми.
97. Повністю відповідано сучасним стандартам та практикам розробки програмного забезпечення.
98. Застосовано найкращі методології тестування та верифікації програмних систем.
99. Реалізовано повний функціональний набір для роботи з геометричними параметрами акваріума.
100. Виконано всі пункти технічного завдання без винятків та компромісів.