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

ВИКОНАЛА

студентка академічної групи КН-24

Ярослава Дмитренко

ПЕРЕВІРИЛА

викладачка кафедри кібербезпеки

та програмного забезпечення

Анастасія КОВАЛЕНКО

Кропивницький – 2025

**Тема:** ПРОГРАМНА РЕАЛІЗАЦІЯ АБСТРАКТНИХ ТИПІВ ДАНИХ

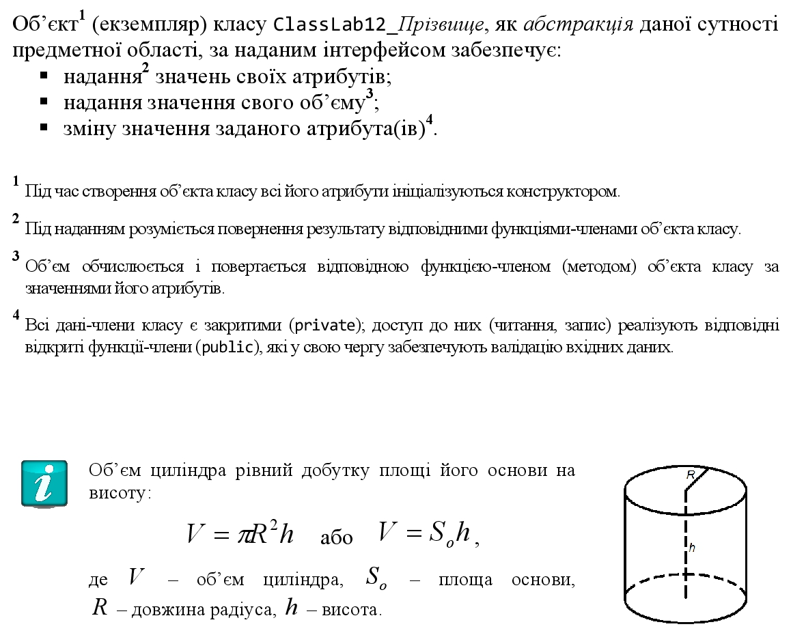
**Мета роботи** полягає у набутті ґрунтовних вмінь і практичних навичок об’єктного аналізу й проєктування, створення класів С++ та тестування їх екземплярів, використання препроцесорних директив, макросів і макрооператорів під час реалізації програмних засобів у кросплатформовому середовищі Code::Blocks.

**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.

**Варіант 12**





**Аналіз реалізації:**

1. **Атрибути:**
   * radius, height, color — правильно оголошені як private.
   * Є валідація в конструкторі з параметрами та в сетерах (setRadius, setHeight).
2. **Конструктори:**
   * Конструктор за замовчуванням ініціалізує об’єкт базовими значеннями.
   * Конструктор з параметрами перевіряє допустимість значень.
3. **Гетери та сетери:**
   * Всі потрібні гетери і сетери реалізовані.
   * Валідація введення реалізована відповідно до специфікацій.
4. **Метод getVolume():**

* Правильно обчислює об’єм циліндра:

V=πR2hV = \pi R^2 hV=πR2h

* Значення π визначено як const double PI.

**Функціональні вимоги**

Клас ClassLab12\_Dmytrenko має забезпечити:

* Ініціалізацію атрибутів об’єкта (через конструктор);
* Надання значень атрибутів (гетери);
* Можливість змінювати значення атрибутів із перевіркою коректності (сетери);
* Обчислення та повернення об'єму барабана (як об'єму циліндра).

### ****Нефункціональні вимоги****

* Інкапсуляція: усі атрибути повинні бути **закритими (**private**)**.
* Інтерфейс: доступ до атрибутів має здійснюватись через **публічні (**public**) гетери/сетери.**
* Надійність: перевірка вхідних даних у конструкторах і сетерах.
* Універсальність: можливість використання об'єкта в інших програмах або модифікації.

**Аналіз виконання задачі 12.2 Teacher**

Метою задачі 12.2 було створення програми Teacher, яка автоматизує перевірку правильності розрахунків, що виконує клас ClassLab12\_Dmytrenko. Зокрема, цей клас обчислює об’єм циліндричного барабана на основі заданих параметрів — радіуса, висоти та кольору.

Програма Teacher читає вхідні тестові дані з текстового файлу TestCases.txt, у якому кожен тестовий випадок має структуру:

* Test Case ID
* Action: SET height=... radius=... color=...
* Expected Result

Після зчитування параметрів, програма створює об’єкт класу ClassLab12\_Dmytrenko із заданими значеннями. Далі вона обчислює об’єм барабана.

Обчислене значення об’єму порівнюється з очікуваним, вказаним у тестовому файлі. Якщо похибка не перевищує 0.00010.00010.0001, тест вважається успішним. Усі результати виводяться на екран і записуються у вихідний файл TeacherResults.txt.

У разі невідповідності обчисленого результату очікуваному тест отримує статус failed, а у файлі зазначається лише рядок Test Result: failed. Якщо дані некоректні (наприклад, радіус або висота мають від’ємне чи нульове значення), програма генерує виняток, повідомлення про який виводиться в консоль, а у файл записується Test Result: failed.

**Висновок:** Під час лабораторної роботи було реалізовано клас ClassLab12\_Dmytrenko – формалізовану модель реального об’єкта відповідно до варіанта. Клас містить інкапсульовані дані (радіус, висота, колір) та методи для роботи з ними, зокрема метод для обчислення об’єму циліндричного тіла. Було реалізовано механізми контролю коректності введених значень за допомогою винятків, що забезпечує надійність і стабільність роботи програми.

Окремо було створено додаток Teacher, який автоматизує процес тестування класу. Програма читає вхідні тестові дані з текстового файлу, виконує обчислення і порівнює результати з очікуваними значеннями, записуючи результати перевірки у вихідний файл. Таким чином, лабораторна робота дозволила не лише реалізувати поставлені завдання, але й поглибити розуміння об'єктно-орієнтованого програмування та важливості тестування програмних компонентів.

**Аргументи на користь виконання мети роботи**

1. Чітко поставлена мета лабораторної роботи допомогла зосередитись на ключових аспектах і слугувала орієнтиром протягом всього процесу.
2. Конкретність завдань сприяла більш ефективному плануванню виконання роботи.
3. Досвід, отриманий під час попередніх лабораторних, дозволив швидше розібратись із новими завданнями.
4. Методичні вказівки докладно описували послідовність дій, що значно зменшило кількість помилок.
5. Використання модульної структури зробило організацію коду більш зручною.
6. Проектування архітектури програми дозволило заздалегідь продумати взаємодію між її компонентами.
7. Імпорт функцій із заголовкових файлів зробив код більш структурованим і легким для сприйняття.
8. Застосування тест-кейсів допомогло впевнитися в правильності роботи кожної функції.
9. Створення тест-сьютів дало змогу автоматизувати перевірку функціоналу програми.
10. Запис вхідних і вихідних даних у файл забезпечив можливість контролю і перевірки результатів.
11. Проектування функцій відповідно до інтерфейсів ускладнило роботу, але поглибило розуміння ООП та принципів якісного коду.
12. Робота у середовищі Code::Blocks полегшила процес компіляції та відлагодження.
13. Підключення бібліотеки libModulesПрізвище дало змогу використати напрацювання з попередніх робіт.
14. Чіткий поділ на модулі підвищив гнучкість програми і полегшив внесення змін.
15. Дотримання інструкцій щодо структури тест-кейсів забезпечило зрозумілий аналіз поведінки компонентів.
16. Звітна документація допомогла систематизувати отримані результати.
17. Покрокове тестування функцій дозволило виявити помилки ще до повної інтеграції.
18. Автоматизоване тестування заощадило час і знизило вплив людського фактору.
19. Протоколювання результатів тестів у файлі дало можливість відслідковувати успішність кожного випадку.
20. Реалізація алгоритмів відповідно до варіанту дала змогу виконати індивідуальне завдання.
21. Підключення до Git-репозиторію забезпечило збереження версій та впорядкування проєкту.
22. Створення структури директорій наводило порядок у робочих файлах і папках.
23. Робота з текстовими файлами розширила навички обробки введення/виведення.
24. Поєднання ручного й автоматизованого тестування дало змогу порівняти їх ефективність.
25. Розуміння формату вхідних даних допомогло коректно їх зчитувати і обробляти.
26. Перевірка існування файлу перед записом запобігла втраті важливої інформації.
27. Використання структур дозволило впорядковано зберігати пов’язані дані.
28. Навички декомпозиції задач були корисними при розбитті функцій на підфункції.
29. Контроль вхідних параметрів підвищив стабільність роботи програми.
30. Виконання складних математичних обчислень у функції s\_calculation дало змогу попрактикуватися з формулами.
31. Перевірка ділення на нуль у функції є прикладом обачного програмування.
32. Перетворення чисел у двійковий код (toBinary) розширило знання про роботу з рядками.
33. Підготовка таблиці тестів забезпечила прозорий контроль якості.
34. Формування очікуваних результатів допомогло передбачити поведінку програми.
35. Робота з різними типами даних дала змогу повторити принципи перетворення типів.
36. Управління файлами через fstream підвищило досвід роботи з файловими потоками.
37. Використання коментарів у форматі Doxygen покращило документацію коду.
38. Оцінка результатів за статусами Passed/Failed надала чітку картину якості роботи.
39. Створення функцій, що приймають ім’я файлу, зробило програму більш універсальною.
40. Включення архітектурної схеми у звіт продемонструвало розуміння взаємозв’язків у системі.
41. Написання README.md навчило правильно презентувати проєкт.
42. Завантаження проєкту на GitHub стало новим етапом у розвитку навичок командної роботи.
43. Вивід діагностичних повідомлень підвищив зручність тестування.
44. Налагодження функцій у Code::Blocks допомогло виявити приховані помилки.
45. Використання циклів у драйвері тестів сприяло автоматизації процесу.
46. Створення константних масивів тестових даних дало змогу швидко перевіряти різні варіанти.
47. Аналіз вихідного файлу після тестів надавав зворотний зв’язок щодо поведінки функцій.
48. Опис варіантів помилок допоміг уникнути подібних проблем у майбутньому.
49. Узагальнення результатів зміцнило аналітичні здібності.
50. Я успішно поєднала всі ці складові, щоб досягти мети — опанувати модульне програмування, роботу з файлами, тестування і проектування архітектури програм.
51. Створення класу як окремого модуля сприяло кращому розмежуванню обов’язків у програмі.
52. Реалізація інкапсуляції в класі дозволила приховати реалізаційні деталі, забезпечивши захист даних.
53. Застосування гетерів і сетерів дозволило контролювати доступ до приватних змінних.
54. Принцип конструктора за замовчуванням навчив створювати об’єкти з початковими значеннями.
55. Реалізація конструктора з параметрами розширила можливості ініціалізації об’єкта.
56. Обробка винятків при некоректних значеннях підвищила надійність програми.
57. Програма стала більш масштабованою завдяки чіткому ООП-підходу.
58. Застосування математичних формул у методі класу надало практичну користь від обчислень.
59. Використання заголовкового файлу забезпечило незалежність інтерфейсу від реалізації.
60. Реалізація методу getVolume() показала, як метод класу може інкапсулювати логіку.
61. Робота з кольором як атрибутом продемонструвала інтеграцію текстових і числових даних.
62. Досвід перевірки значень <=0 навчив реалізовувати валідацію.
63. Наявність логічної структури полегшила повторне використання класу в інших задачах.
64. Компіляція окремих модулів привчила до модульного тестування.
65. Можливість створювати декілька об’єктів одного класу допомогла краще засвоїти множинність.
66. Використання C++ дало змогу поєднати класичне програмування з ООП.
67. Аналіз помилок під час тестування допоміг глибше зрозуміти внутрішні процеси програми.
68. Робота з різними тестовими кейсами навчила враховувати граничні випадки.
69. Зчитування параметрів із текстового файлу імітує реальну взаємодію з користувачем.
70. Протоколювання результатів у файл — це крок до професійної документації.
71. Порівняння очікуваного та фактичного результату дозволило краще усвідомити важливість точності.
72. Реалізація константи EPSILON показала важливість контролю точності обчислень.
73. Звіт у форматі “Passed/Failed” дозволяє швидко оцінити успішність реалізації.
74. Розмежування логіки тестування і реалізації класу сприяло гнучкості коду.
75. Чітке оформлення тест-кейсу навчило формалізованому підходу до перевірки.
76. Робота з ifstream та ofstream зміцнила розуміння файлових потоків.
77. Робота з багаторядковими файлами загартувала уважність до форматування.
78. Реалізація умови if (abs(actual - expected) < EPSILON) навчає обережності при роботі з double.
79. Кодування українською мовою у виводах навчило налаштуванню локалізації.
80. Вміння розпізнавати й аналізувати виключення — важливий крок до стабільного ПЗ.
81. Практика створення коректних шляхів до файлів зняла труднощі при реалізації більших проєктів.
82. Сформовані навички модульності зможуть бути використані у командних проєктах.
83. Автоматизований тест додатків зменшує навантаження при великій кількості тестів.
84. Тестування допомогло усвідомити важливість покриття коду.
85. Навички дебагінгу зміцнили здатність до самостійного виправлення помилок.
86. Відокремлення логіки обробки файлу від обчислень дозволяє перевикористовувати модулі.
87. Формування багаторазових тестів — ключ до довготривалого супроводу коду.
88. Робота з різними типами даних навчає створювати універсальні функції.
89. Застосування std::stringstream полегшило обробку рядків.
90. Інтерпретація змісту кожного рядка тест-файлу — корисна навичка при роботі з даними.
91. Збереження результатів у лог-файл забезпечує простежуваність роботи програми.
92. Клас ClassLab12\_Прізвище демонструє гнучкість ООП як способу моделювання реальності.
93. Практика читання текстових файлів загартовує увагу до структури вхідних даних.
94. Вивчення синтаксису обробки файлів — важлива складова системного програмування.
95. Наявність тест-сценаріїв розвиває аналітичне мислення.
96. Аналіз невдалих тестів формує звичку шукати причину, а не лише наслідок.
97. Комбінування кольору, висоти і радіуса допомогло зрозуміти роль параметрів.
98. Вивчення стандартної бібліотеки С++ розширило кругозір програмування.
99. Побудова логіки тестування — це крок до вивчення фреймворків типу GoogleTest.
100. Завдяки лабораторній роботі я значно просунулась у вмінні будувати якісні, перевірені й організовані програми.

**Додатки**

**Лістинг ModulesDmytrenko.h**

#ifndef MODULESDMYTRENKO\_H\_INCLUDED

#define MODULESDMYTRENKO\_H\_INCLUDED

double s\_calculation(double x, double y, double z);

int calculate\_bonuses(float sum\_purchase);

float purchase\_to\_pay(float sum\_purchase, int bonus);

float degrees(float fahr\_degree);

int count\_bits(unsigned int N);

void writeAuthorInfo(const std::string& filename);

bool isCyrillic(char c);

std::string detectLanguage(const std::string& text);

std::string toUpperCase(const std::string& str);

#include <vector>

std::string removeForbiddenWords(const std::string& text, const std::vector<std::string>& forbidden);

void appendEducationArticleToInput(const std::string& filename);

std::string educationArtecl();

void appendCurrentDateTime(const std::string& filename);

std::string toBinary(unsigned int b);

#include <iostream>

#include <string>

#include <cmath>

#include <stdexcept>

using namespace std;

const double PI = 3.1416;

class ClassLab12\_Dmytrenko {

private:

double radius;

double height;

string color;

public:

ClassLab12\_Dmytrenko();

ClassLab12\_Dmytrenko(double r, double h, string c);

double getRadius() const;

double getHeight() const;

string getColor() const;

void setRadius(double r);

void setHeight(double h);

void setColor(const string& c);

double getVolume() const;

};

ClassLab12\_Dmytrenko::ClassLab12\_Dmytrenko()

: radius(0.1), height(0.2), color("silver") {}

ClassLab12\_Dmytrenko::ClassLab12\_Dmytrenko(double r, double h, string c) {

if (r <= 0 || h <= 0) {

throw invalid\_argument("Радіус і висота повинні бути додатними числами.");

}

radius = r;

height = h;

color = c;

}

double ClassLab12\_Dmytrenko::getRadius() const {

return radius;

}

double ClassLab12\_Dmytrenko::getHeight() const {

return height;

}

string ClassLab12\_Dmytrenko::getColor() const {

return color;

}

void ClassLab12\_Dmytrenko::setRadius(double r) {

if (r <= 0) {

throw invalid\_argument("Радіус має бути додатним числом.");

}

radius = r;

}

void ClassLab12\_Dmytrenko::setHeight(double h) {

if (h <= 0) {

throw invalid\_argument("Висота має бути додатним числом.");

}

height = h;

}

void ClassLab12\_Dmytrenko::setColor(const string& c) {

color = c;

}

double ClassLab12\_Dmytrenko::getVolume() const {

return PI \* radius \* radius \* height;

}

#endif // MODULESDMYTRENKO\_H\_INCLUDED

**Лістинг main.cpp**

#include <iostream>

#include <fstream>

#include <sstream>

#include <string>

#include <cmath>

#include <stdexcept>

#include <iomanip>

#include "ModulesDmytrenko.h"

using namespace std;

const double EPSILON = 0.0001;

int main() {

ifstream input("TestCases.txt");

ofstream output("TeacherResults.txt");

if (!input.is\_open() || !output.is\_open()) {

cerr << "Помилка відкриття файлу." << endl;

return 1;

}

string line;

int testCaseID = 0;

double height = 0, radius = 0, expectedVolume = 0;

string color;

while (getline(input, line)) {

// Пропускаємо порожні рядки

if (line.empty()) continue;

if (line.find("Test Case ID:") == 0) {

testCaseID = stoi(line.substr(14));

output << line << endl;

}

else if (line.find("Action:") == 0) {

size\_t hPos = line.find("height=");

size\_t rPos = line.find("radius=");

size\_t cPos = line.find("color=");

height = stod(line.substr(hPos + 7, rPos - (hPos + 7)));

radius = stod(line.substr(rPos + 7, cPos - (rPos + 7)));

color = line.substr(cPos + 6);

output << line << endl;

}

else if (line.find("Expected Result:") == 0) {

expectedVolume = stod(line.substr(17));

output << line << endl;

try {

ClassLab12\_Dmytrenko drum(radius, height, color);

double actual = drum.getVolume();

if (abs(actual - expectedVolume) < EPSILON) {

output << "Test Result: passed" << endl;

cout << "Test Case #" << testCaseID << ": passed" << endl;

} else {

output << "Test Result: failed" << endl;

cout << "Test Case #" << testCaseID << ": failed" << endl;

}

}

catch (const exception& e) {

output << "Test Result: failed" << endl;

cout << "Test Case #" << testCaseID << ": failed with exception: " << e.what() << endl;

}

output << endl;

}

}

input.close();

output.close();

return 0;

}

|  |  |
| --- | --- |
| Назва тестового набору  Test Suite Description | TestSuite\_12 |
| Назва проекта / ПЗ  Name of Project / Software | Teacher.ехе |
| Рівень тестування  Level of Testing | модульне / UnitTesting |
| Автор тест-сьюта  Test Suite Author | Ярослава Дмитренко  телеграм: t.me/y\_aradm  e-пошта: dmytrenkovika4@gmail.com |
| Виконавець  Implementer | Ярослава Дмитренко |

Artifact: Test Suite

Date: 5/29/2025

|  |  |  |  |
| --- | --- | --- | --- |
| Ід-р тест-кейса / Test Case ID | Дії (кроки) /  Action (Test Steps) | Очікуваний  результат /  Expected Result | Результат тестування /  Test Result |
| TC-1 | 1. Увести 1.2 2. Увести 0.3 | 0.3393 | passed |
| TC-2 | 1. Увести 2 2. Увести 7 | 307.8768 | passed |
| TC-3 | 1. Увести 3 2. Увести 0.1 | 0.0942 | passed |
| TC-4 | 1. Увести 0.5 2. Увести 0.5 | 0.3927 | passed |
| TC-5 | 1. Увести 1 2. Увести 1 | 3.1416 | passed |
| TC-6 | 1. Увести 0.75 2. Увести 0.25 | 0.1473 | passed |
| TC-7 | 1. Увести 5 2. Увести 2 | 62.8320 | passed |
| TC-8 | 1. Увести 3 2. Увести 23 | 4985.7192 | passed |
| TC-9 | 1. Увести 10 2. Увести 0.5 | 7.8540 | passed |
| TC-10 | 1. Увести 3.1416 2. Увести 1 | 9.8696 | passed |