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

Марина Середа

ПЕРЕВІРИЛА

викладачка кафедри кібербезпеки

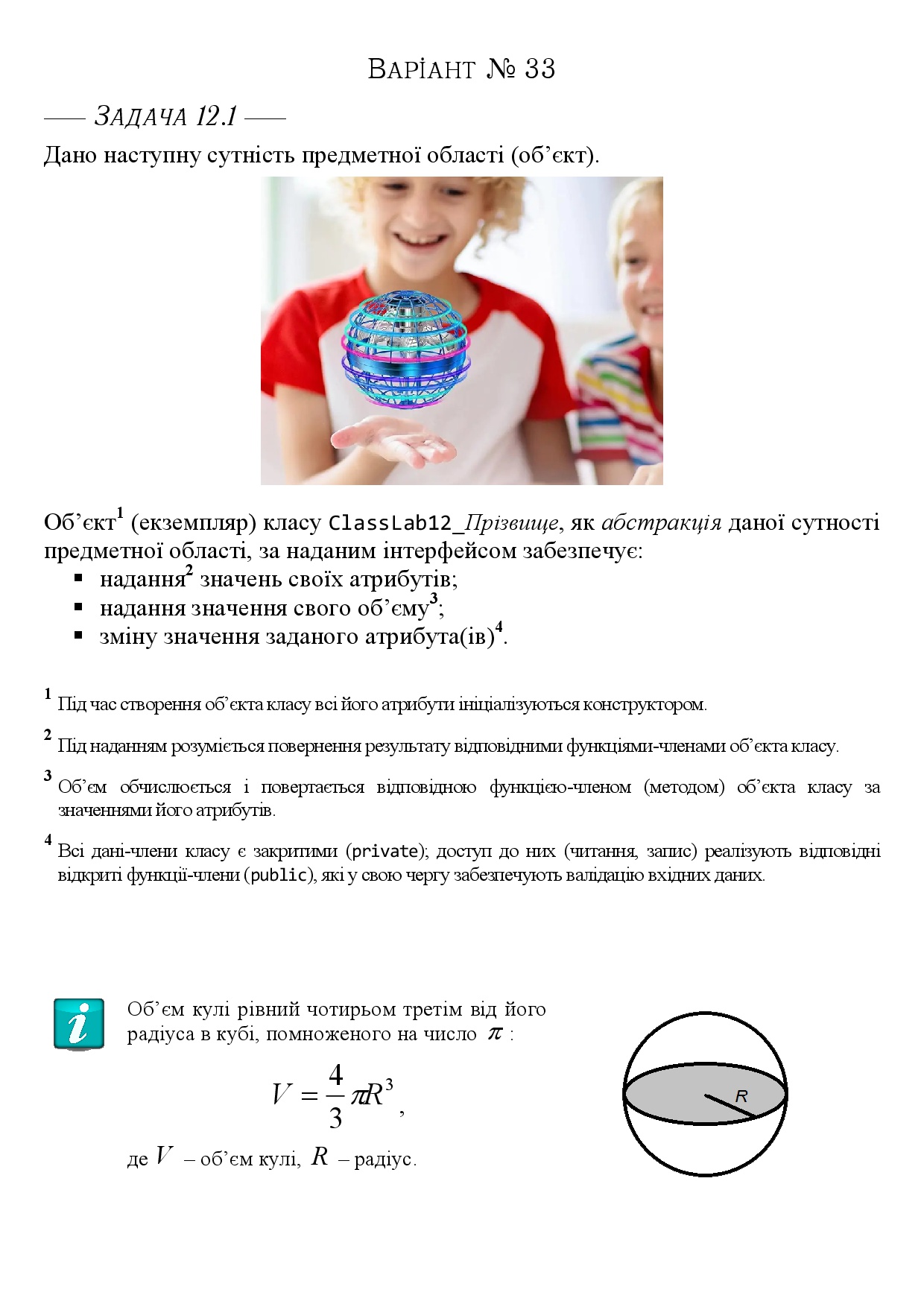
та програмного забезпечення

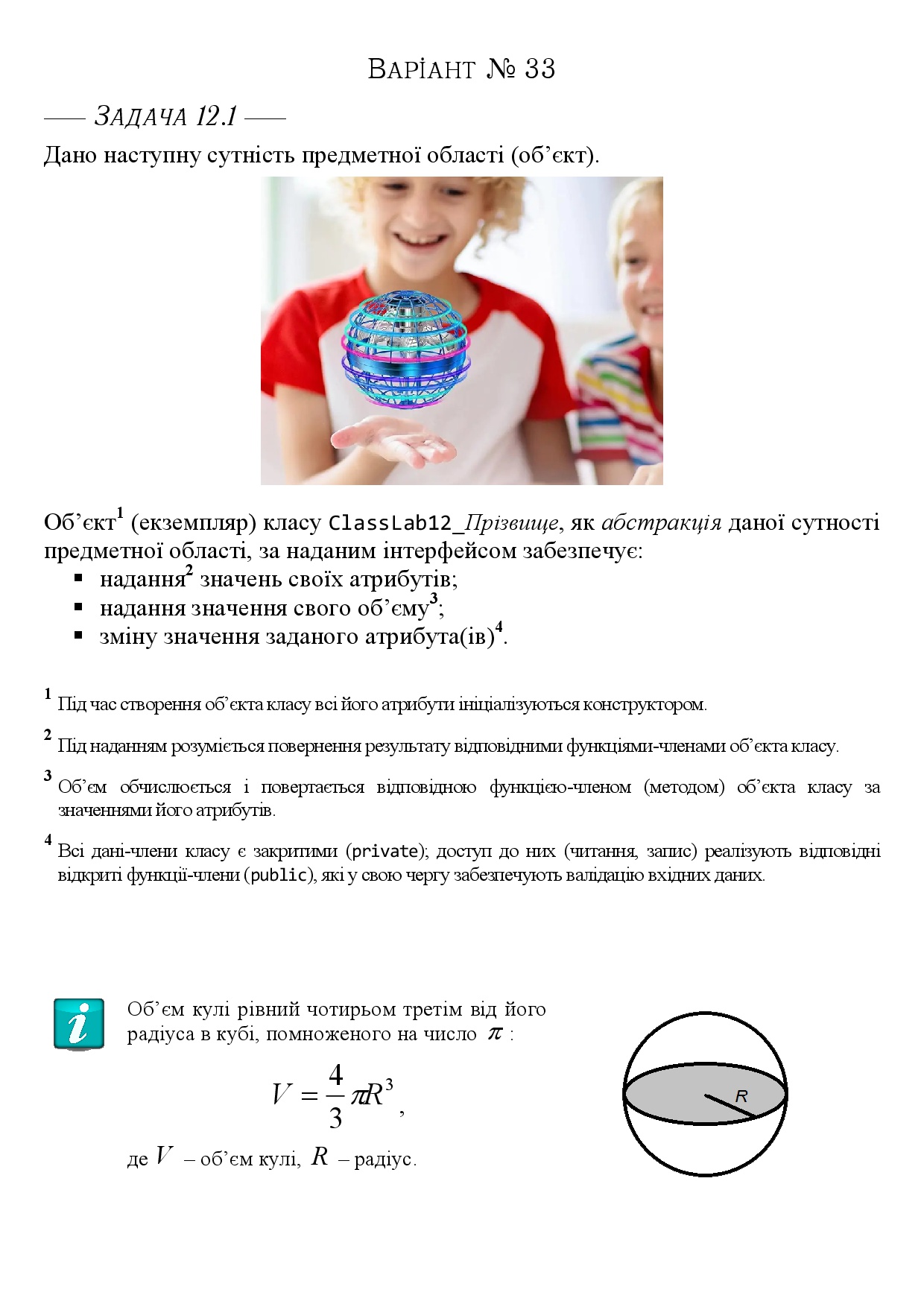
Анастасія КОВАЛЕНКО

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 куля (візуалізована у вигляді об'єкта, що тримає дитину на фото).

**Об'єктний аналіз**

Клас: ClassLab12\_Sereda

Атрибут: radius (радіус кулі)

Методи:

* setRadius(float) — встановлення значення атрибута
* getRadius() — значення атрибута
* getVolumeBall() — розрахунок об'єму кулі за формулою:

V = (4/3)\*π\*R³

**Визначення інтерфейсів сутності предметної області**

* Інтерфейс надання значень атрибутам — реалізовано методом setRadius()
* Інтерфейс доступу до значень атрибутів — реалізовано методом getRadius()
* Інтерфейс доступу до значення об'єму (розрахунок) — реалізовано методом getVolumeBall()
* Ініціалізація — через конструкторClassLab12\_Sereda(float radius)

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

Аналіз до вимог програмного модуля ClassLab12\_Sereda

### ****Функціональні вимоги****

Програмний модуль має можливість реалізовувати об’єкт кулі як екземпляр класу такими ознаками:

#### Ініціалізація об'єкта:

* Конструктор без параметрів:ClassLab12\_Sereda()
* Конструктор з параметром:ClassLab12\_Sereda(float radius)
* Ініціалізує атрибут radius при створенні об'єкта.

#### Доступ до атрибутів:

* **Читання:** getRadius() — повертає значення радіуса.
* **Запис:** setRadius(float) — встановлює нове значення радіуса.

1. Додаткова функціональність:

* getVolumeBall()— об'єм кулі обчислює за формулою:

V = (4/3)\*π\*R³

**Нефункціональні вимоги**

#### Інкапсуляція:

* Атрибут radius оголошено як private— доступ лише через відкриті методи.

#### Безпека / Надійність:

* Потенційно слід додати перевірку у setRadius() (наприклад, не допускати від'ємних значень).

1. Простота та читабельність:

* Інтерфейс модуля зрозумілий, усі методи забезпечують одну чітку функцію.

#### Математична коректність:

* Формула для об'єму реалізована через M\_PIі pow() з бібліотеки <cmath>, що забезпечує точність обчислень.

**Вимоги до інтерфейсу**

* Відповідає принципам ООП: інкапсуляція, інтерфейси доступу (get, set) та обчислення (getVolumeBall()).
* Структура класу дозволяє інтеграцію в більші системи моделювання об'єктів у 3D-просторі або освітні симуляції.

### ****Архітектурна****

#### **Конструктори**

* **За замовчуванням:** дозволяє створити кулі з нульовим радіусом.
* **Параметризований:** дозволяє створювати об'єкта з відразу заданим значенням радіуса.

#### **Атрибут**

* Radius — приватна змінна, що інкапсулює фізичний параметр кулі.

#### **Сетери/Гетери**

* setRadius(float) — встановлює значення радіуса з можливою валідацією.
* getRadius() — повертає поточне значення.

#### **Функціональний метод**

* getVolumeBall() — реалізує формулу об'єму кулі.

**Завдання 2**

### ****Аналіз вимог до ПЗ Учитель****

### ****Функціональні вимоги:****

ПЗ обов'язково:

* Отримувати від користувача кількість тестів.
* Зчитувати параметри кожного тесту:
  + - радіус кулі;
    - очікуваний об'єм;
* Створювати об'єкт класу ClassLab12\_Sereda з переданим радіусом;
* обчислювати фактичний об'єм кулі за формулою;
* Порівнювати обчислений об'єм із очікуваним (з похибкою);
* Виведіть результат у файл (test\_results.txt).

#### **Нефункціональні вимоги:**

* Надійність: перевірка на помилки відкриття файлу.
* Точність: обчислення з точністю до 5 знаків після коми.
* Кросплатформенність: використання стандартної бібліотеки C++.

Вхідні дані:

int – testCount

float – inputRadius

float – expectedVolume

Вихідні дані:

float – computedVolume

string – (рядок, який записується у файл)

**Висновок:** у ході лабораторної роботи №12, я набула 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.

**Аргументи**

1. Забезпечує практичне застосування теоретичних знань з програмування.
2. Формує навички розробки програмного забезпечення на C++.
3. Поглиблює розуміння принципів об’єктно-орієнтованого програмування.
4. Розвиває аналітичне мислення.
5. Навчає концептуалізації предметної області.
6. Дає досвід структурування даних у вигляді класів.
7. Навчає використовувати конструктори класів.
8. Удосконалює навички інкапсуляції та використання гетерів/сетерів.
9. Формує розуміння абстрактних типів даних.
10. Розвиває вміння створювати багаторазово використовуваний код.
11. Підвищує навички модульного програмування.
12. Дає розуміння структури заголовкових файлів у C++.
13. Навчає роботі з препроцесорними директивами.
14. Сприяє розвитку вмінь логічного контролю умов компіляції.
15. Формує навички компіляції проєктів у середовищі Code::Blocks.
16. Ознайомлює зі структурою проектів у C++.
17. Поглиблює розуміння математичних формул у програмуванні.
18. Дає практичний досвід з обчислення об’єму кулі.
19. Навчає ефективному застосуванню математичних функцій з <cmath>.
20. Готує до вирішення практичних завдань на технічних спеціальностях.
21. Реалізація класу ClassLab12\_Sereda відповідає принципам ООП.
22. Створення власних типів даних дозволяє краще моделювати реальні об’єкти.
23. Конструктори демонструють гнучкість ініціалізації об’єктів.
24. Валідація даних (наприклад, радіуса) покращує стабільність коду.
25. Розділення інтерфейсу та реалізації — ознака якісного коду.
26. Код можна легко масштабувати (наприклад, для кулі іншої форми).
27. Програмна реалізація відповідає сучасним стандартам C++.
28. Код легко читається, що важливо для командної роботи.
29. Всі методи мають чітко визначену відповідальність.
30. Реалізовано перевантаження конструкторів — показовий приклад гнучкості.
31. Метод getVolumeBall() правильно реалізує математичну формулу.
32. Стандарти стилю коду підтримано — зрозумілий синтаксис.
33. Код дає змогу протестувати функціональність через об’єкти.
34. Реалізація коду допомагає краще зрозуміти роль інкапсуляції.
35. Наявність #ifndef запобігає багатократному включенню заголовку.
36. Власна бібліотека демонструє розширюваність коду.
37. Програмне рішення є частиною реального проєкту — файл Teacher.cpp.
38. Існує реальний механізм зчитування і обробки тестів.
39. Результати тестування протоколюються у файл — звітність.
40. Можна перевірити правильність розрахунків на великій кількості кейсів. Лабораторна містить реалізацію unit-тестування.
41. Користувач сам вводить тестові параметри.
42. Результати тестування зберігаються у текстовий файл.
43. Вказано точність виведення результатів до 5 знаків.
44. Передбачено перевірку коректності обчислень.
45. Створено реальний тестовий сценарій.
46. Можна оцінити похибку обчислень.
47. Можна перевірити стабільність програми при різних значеннях.
48. Дає навички написання тест-сценаріїв.
49. Сприяє навчанню базовому рівню автоматизованого тестування.
50. Результати логуються для подальшого аналізу.
51. Реалізовано умову проходження тесту на основі точності.
52. Можна аналізувати кількість успішних та провальних тестів.
53. Наявність тест-сьютів спонукає до повторюваного тестування.
54. Дає можливість тестувати критичні граничні значення (наприклад, 0).
55. Дає уявлення про ефективність коду.
56. Навчає документуванню тестів.
57. Дає змогу зберігати історію тестування.
58. Підвищує відповідальність за якість реалізованої функції.
59. Формує навички аналізу помилок.
60. Закріплює дотримання принципів чистого коду.
61. Формує звичку до розмежування логіки та інтерфейсу (заголовкові та cpp-файли).
62. Демонструє переваги чіткого іменування змінних та методів.
63. Вчить розробляти код, який легко підтримувати і масштабувати.
64. Показує важливість дотримання стандартів кросплатформенності (Code::Blocks, C++ стандарт).
65. Формує навички створення unit-тестів.
66. Пояснює важливість точності при обчисленнях.
67. Навчає запису результатів тестування у файл.
68. Розвиває вміння перевіряти похибку обчислень.
69. Дає змогу реалізувати систему Pass/Fail у тестах.
70. Навчає роботі з файлами вводу/виводу.
71. Пояснює значення перевірки відкриття файлів (ofstream).
72. Показує значення логування результатів.
73. Навчає форматуванню виводу через setprecision, fixed.
74. Пояснює як виводити результати у вигляді зрозумілого звіту.
75. Формує вміння працювати з препроцесорними директивами #include, #define.
76. Дає досвід створення заголовкових файлів (.h).
77. Навчає дотримуватись структури програмного проєкту.
78. Поглиблює розуміння простору імен (namespace).
79. Тренує написання функцій з параметрами і поверненням значень.
80. Дає практику використання математичних бібліотек, як-от <cmath>.
81. Формує вміння працювати з математичними функціями pow, M\_PI.
82. Пояснює реалізацію формул у вигляді функцій.
83. Навчає обчислювати об’єм кулі з використанням реальних даних.
84. Показує важливість перевірки коректності результатів.
85. Закладає фундамент для роботи з більш складними геометричними структурами.
86. Допомагає зрозуміти принцип повторного використання коду в інших проєктах.
87. Дає змогу на практиці опрацювати концепцію “одна задача – один клас”.
88. Навчає думати як розробник, а не просто як користувач програми.
89. Допомагає зрозуміти структуру та логіку сучасних програмних бібліотек.
90. Пояснює, як грамотно взаємодіють основні компоненти ПЗ: класи, функції, об'єкти, тести.
91. Надає досвід самостійного виявлення та виправлення помилок.
92. Підвищує навички логічної декомпозиції завдань.
93. Учитись читати, аналізувати і розуміти чужий код.
94. Дає приклад правильного оформлення програмного звіту.
95. Пояснює важливість назв файлів, структури папок (\Lab12\prj\, \TestSuite\) у розробці.
96. Забезпечує приклад практичного застосування принципу DRY (Don’t Repeat Yourself).
97. Демонструє як пов’язати кілька модулів у єдиний програмний продукт.
98. Пояснює цінність добре документованого коду.
99. Дає змогу практикуватися в застосуванні англійської мови для назв змінних і методів.
100. Розвиває навички зберігання результатів у зовнішніх текстових файлах.

**Додатки**

|  |  |
| --- | --- |
| Назва тестового набору  Test Suite Description | TS\_Teacher |
| Назва проекта / ПЗ  Name of Project / Software | ModulesSereda |
| Рівень тестування  Level of Testing | модульне / Unit Testing |
| Автор тест-сьюта  Test Suite Author | Середа Марина |
| Виконавець  Implementer | Середа Марина |

|  |  |  |  |
| --- | --- | --- | --- |
| Ід-р тест-кейса / Test Case ID | Дії (кроки) /  Action  (Test Steps) | Очікуваний  результат /  Expected Result | Результат тестування (пройшов/не вдалося/ заблокований) /  Test Result (passed/failed/ blocked) |
| 1 | 1. Запуск програми. 2. Увести 0.00000. 3. Увести 0.00000. | Обчислений об'єм: 0.00000  Результат тестування: Passed | Passed |
| 2 | 1. Увести 1.00000. 2. Увести 4.18879. | Обчислений об'єм: 4.18879  Результат тестування: Passed | Passed |
| 3 | 1. Увести 2.00000. 2. Увести 33.51032. | Обчислений об'єм: 33.51032  Результат тестування: Passed | Passed |
| 4. | 1. Увести 0.50000. 2. Увести 0.52360. | Обчислений об'єм: 0.52360  Результат тестування: Passed | Passed |
| 5. | 1. Увести 10.00000. 2. Увести 4188.79004. | Обчислений об'єм: 4188.79004  Результат тестування: Passed | Passed |
| 6. | * 1. Увести 0.00001.   2. Увести 0.00000. | Обчислений об'єм: 0.00000  Результат тестування: Passed | Passed |
| 7. | 1. Увести 3.14159. 2. Увести 129.87848. | Обчислений об'єм: 129.87848  Результат тестування: Passed | Passed |
| 8. | 1. Увести 1000.00000.  2. Увести 4188790272.00000. | Обчислений об'єм: 4188790272.00000  Результат тестування: Passed | Passed |
| 9. | 1. Увести 1.23400. 2. Увести 7.87108. | Обчислений об'єм: 7.87108  Результат тестування: Passed | Passed |
| 10. | 1. Увести 5.67800. 2. Увести 766.78705. | Обчислений об'єм: 766.78705  Результат тестування: Passed | Passed |

**Лістинг**

**ModulesSereda.h**

#ifndef MODULESSEREDA\_H\_INCLUDED

#define MODULESSEREDA\_H\_INCLUDED

#include <cmath>

class ClassLab12\_Sereda{

public:

ClassLab12\_Sereda(){}

ClassLab12\_Sereda(float radius);

void setRadius(float radius);

float getRadius();

float getVolumeBall();

private:

float radius = 0.0;

};

ClassLab12\_Sereda::ClassLab12\_Sereda(float radius){

this->radius = radius;

}

void ClassLab12\_Sereda::setRadius(float radius){

this->radius = radius;

}

float ClassLab12\_Sereda::getRadius(){

return radius;

}

float ClassLab12\_Sereda::getVolumeBall(){

return (4 \* M\_PI \* pow(radius, 3))/3;

}

#endif // MODULESSEREDA\_H\_INCLUDED

**Teacher.cpp**

#include <iostream>

#include <fstream>

#include <iomanip>

#include <cmath>

#include "ModulesSereda.h"

using namespace std;

int main() {

ofstream outfile("test\_results.txt");

if (!outfile) {

cerr << "Не вдалося відкрити файл для запису." << endl;

return 1;

}

int testCount;

cout << "Скільки тест-кейсів ви хочете виконати? ";

cin >> testCount;

outfile << fixed << setprecision(5);

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

float inputRadius, expectedVolume;

cout << "\nТест #" << i + 1 << endl;

cout << "Введіть радіус кулі: ";

cin >> inputRadius;

cout << "Введіть очікуваний об'єм кулі: ";

cin >> expectedVolume;

ClassLab12\_Sereda ball(inputRadius);

float computedVolume = ball.getVolumeBall();

// Запис до файлу

outfile << "Тест #" << i + 1 << ":\n";

outfile << "Радіус: " << inputRadius << "\n";

outfile << "Очікуваний об'єм: " << expectedVolume << "\n";

outfile << "Обчислений об'єм: " << computedVolume << "\n";

outfile << "Результат тестування: "

<< (fabs(computedVolume - expectedVolume) < 0.0001 ? "Passed" : "Failed") << "\n\n";

}

outfile.close();

cout << "\nУсі результати тестування збережено у файл test\_results.txt\n";

return 0;

}