Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

Механіко-технологічний факультет

ЗВІТ

ПРО ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ № 8

з навчальної дисципліни

“Базові методології та технології програмування”

РЕАЛІЗАЦІЯ СТАТИЧНИХ БІБЛІОТЕК МОДУЛІВ

ЛІНІЙНИХ ОБЧИСЛЮВАЛЬНИХ ПРОЦЕСІВ

ЗАВДАННЯ ВИДАВ

доцент кафедри кібербезпеки та програмного забезпечення

Доренський О. П.

[https://github.com/odorenskyi/](https://github.com/odorenskyi/Dmytro-Parkhomenko-KB18)

ВИКОНАВ

студент академічної групи КБ-24

Марченко К. О.

ПЕРЕВІРИВ

викладач кафедри кібербезпеки   
та програмного забезпечення

Коваленко А. С.

Кропивницький – 2025

**Тема**: Реалізація статичних бібліотек модулів лінійних обчислювальних процесів

**Мета роботи** полягає у набутті ґрунтовних вмінь і практичних навичок застосування теоретичних положень методології модульного програмування, реалізації метода функціональної декомпозиції задач, метода модульного (блочного) тестування, представлення мовою програмування С++ даних скалярних типів, арифметичних і логічних операцій, потокового введення й виведення інформації, розроблення програмних модулів та засобів у кросплатформовому середовищі Code::Blocks (GNU GCC Compiler).

**Завдання до лабораторної роботи**:

1. Реалізувати статичну бібліотеку модулів libModulesMarchenko C/C++, яка містить функцію розв’язування задачі 8.1.
2. Реалізуватии програмне забезпечення розв’язування задачі 8.2 — консольний застосунок.

**Аналіз і постановка задачі 8.1:**

Функціональні вимоги

* Вхідні дані: три числових значення – x, y, z.
* Обчислення: обчислення S за формулою
* Вихід: Результат обчислення S, який має бути коректним при виконанні операцій над дійсними числами.

Обмеження і валідація

* Ділення на нуль: Вираз x/|y − z| → вимагає, щоб |y − z| ≠ 0.
* Вираз (x² + z) / √(|z + x|⋅y) → вимагає, щоб вираз під квадратним коренем (|z + x|⋅y) > 0.
* Вираз cos(x⋅z²)/(2y + π) → знаменник не повинен дорівнювати нулю.

Область визначення функції:

* Вираз під коренем має бути невід’ємним, інакше операція квадратного кореня не може бути виконана над дійсними числами.

Результати тестування s\_calculation зі статичної бібліотеки ModulesMarchenko.а тестовим драйвером:

Test case #1: s\_calculation(2, 3, 1) = 2.64687 == 2.64687 --> passed

Test case #2: s\_calculation(2, 3, 3) = nan == nan --> passed

Test case #3: s\_calculation(0, 4, -1) = nan == nan --> passed

Test case #4: s\_calculation(3, 10, 2) = 2.25376 == 2.25376 --> passed

Test case #5: s\_calculation(1, 0, 2) = nan == nan --> passed

Вихідний код проєкту ModulesMarchenko:

#include <cmath>

#include <stdexcept>

const double EPSILON = 1e-9;

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

// Перевірка для виразу |y-z| (щоб уникнути ділення на 0)

if (std::fabs(y - z) < EPSILON) {

return NAN;

}

// Обчислення першого терму: (x/|y-z|)^3

double term1 = std::pow(x / std::fabs(y - z), 3);

// Перевірка для виразу під квадратним коренем у знаменнику другого терму:

// √(|z+x| \* y) - перевіряємо, що y > 0 та |z+x| > 0

if (y <= 0 || std::fabs(z + x) < EPSILON) {

return NAN;

}

double sqrtDenom = std::sqrt(std::fabs(z + x) \* y);

double term2 = (x \* x + z) / sqrtDenom;

// Перевірка знаменника для третього терму: 2\*y + π

double denominator = 2 \* y + M\_PI;

if (std::fabs(denominator) < EPSILON) {

return NAN;

}

double term3 = std::cos(x \* z \* z) / denominator;

// Обчислення виразу під квадратним коренем

double innerExpression = term1 + term2 - term3;

if (innerExpression < 0) {

return NAN;

}

// Обчислення кінцевого значення S

double S = 1 + std::sqrt(innerExpression);

return S;

}

Вихідний код проєкту TestDriver:

#include <iostream>

#include <sstream>

#include <iomanip>

#include <cmath>

#include "ModulesMarchenko.h"

using namespace std;

string doubleToString(double value, int precision = 5) {

ostringstream oss;

oss << fixed << setprecision(precision) << value;

return oss.str();

}

void test\_s\_calculation() {

double x[5] = {2, 2, 0, 3, 1};

double y[5] = {3, 3, 4, 10, 0};

double z[5] = {1, 3, -1, 2, 2};

double expected\_results[5] = {2.64687, NAN, NAN, 2.25376, NAN};

for (short i = 0; i < 5; i++) {

double result = s\_calculation(x[i], y[i], z[i]);

string resultStr = doubleToString(result);

string expectedStr = doubleToString(expected\_results[i]);

bool passed = false;

if (std::isnan(result) && std::isnan(expected\_results[i])) {

passed = true;

} else if (resultStr == expectedStr) {

passed = true;

}

cout << "Test case #" << i+1 << ": s\_calculation("

<< x[i] << ", " << y[i] << ", " << z[i] << ") = "

<< resultStr << " == " << expectedStr

<< " --> " << (passed ? "passed" : "failed") << endl;

}

}

int main() {

test\_s\_calculation();

return 0;

}

## Аналіз та постановка задачі 8.2

Задача 8.2 складається з трьох підзадач, які виконуються послідовно після введення чисел та символів:

**8.2.1:** Виведення прізвища та імені розробника з використанням символу «©».

**8.2.2:** Обчислення логічного виразу

з виведенням результату у числовому вигляді (1 або 0).

**8.2.3:** Виведення чисел x, y, z в десятковій та шістнадцятковій системах числення, а також обчислення значення S за допомогою функції s\_calculation(), яка знаходиться у заголовковому файлі ModulesMarchenko.h.

### Функціональні вимоги

* **Вхідні дані:**
  + Три числових значення x, y, z (можуть бути як цілі, так і дійсні).
  + Два символи a та b (їх значення використовуються у логічному виразі).
* **Вивід:**
  + Результат кожної підзадачі має бути виведений у стандартний потік виводу cout за допомогою оператора вставки <<.
  + Результати мають включати:
    1. Прізвище та ім'я розробника з символом «©».
    2. Результат логічного виразу у вигляді 1 (true) або 0 (false).
    3. Значення x, y, z у десятковій та шістнадцятковій формах; обчислене значення S.

### Обмеження та перевірки

* Забезпечити коректність введення числових значень і символів.
* Обчислення абсолютного значення та порівняння з модифікованим значенням b.
* Форматування даних для числових систем (десяткова та шістнадцяткова).

Аргументи досягнення мети:

1. Лабораторна робота дозволяє зрозуміти принципи поділу програми на незалежні модулі, що покращує підтримку та масштабованість ПЗ.
2. Створення статичних бібліотек забезпечує повторне використання коду та сприяє ефективній організації проекту.
3. Розбиття задачі на менші функції дозволяє спростити розробку та тестування окремих частин коду.
4. Практична робота з мовою C++ розширює розуміння синтаксису, особливостей компіляції та управління пам’яттю.
5. Знайомство з сучасним середовищем розробки сприяє підвищенню ефективності програмування та налагодження коду.
6. Ведення версійного контролю дає можливість ефективно управляти змінами в проекті та сприяє командній роботі.
7. Детальне оформлення звіту (аналітична частина, архітектурне рішення, тест-кейси) забезпечує ясність і зрозумілість розробленої системи.
8. Розробка контрольних прикладів (unit testing) сприяє виявленню помилок на ранніх етапах розробки.
9. Набір тест-кейсів для системного тестування допомагає перевірити інтеграцію різних модулів в єдиний робочий додаток.
10. Ведення журналу тестування дозволяє простежити історію виконання тестів і швидко локалізувати помилки.
11. Реалізація обчислювальних процесів покращує розуміння роботи з математичними функціями та бібліотеками.
12. Робота з STL-потоками (cout, cin) сприяє засвоєнню принципів обробки даних та їх форматування.
13. Опанування структурування коду через розділення декларацій і визначень сприяє зручності повторного використання компонентів.
14. Збірка статичної бібліотеки допомагає зрозуміти, як відбувається інтеграція окремих модулів у єдине ПЗ.
15. Робота з операціями додавання, множення, ділення та коренем поглиблює розуміння числових обчислень.
16. Аналіз і реалізація алгоритмів для розв’язання задач сприяє розвитку алгоритмічного мислення.
17. Чітка організація коду допомагає уникнути хаотичної структури та забезпечує кращу читаність коду.
18. Реалізація логічних виразів покращує розуміння умовних конструкцій та порівняльних операцій.
19. Використання обробки помилкових ситуацій (exception handling) забезпечує надійність роботи ПЗ.
20. Дотримання вимог ISO/IEC та стандартів оформлення допомагає сформувати звички якісного програмування.
21. Використання маніпуляторів (setw, setprecision, hex) покращує вміння форматувати вивід інформації.
22. Реалізація алгоритмів з використанням циклів і умов забезпечує засвоєння основ логіки програмування.
23. Аналіз ролі функції main поглиблює розуміння початкової точки входу в програму.
24. Практична робота сприяє розвитку навичок відлагодження та пошуку логічних помилок.
25. Хоча лабораторна робота фокусується на модулях, розуміння принципів ООП сприяє подальшому розвитку в програмуванні.
26. Аналіз вимог і постановка задач вчать логічно та послідовно підходити до вирішення складних проблем.
27. Робота з власноруч створеними бібліотеками сприяє розумінню механізмів їх інтеграції у проекти.
28. Розробка тест-кейсів допомагає систематизувати підхід до перевірки працездатності окремих компонентів.
29. Використання Code::Blocks та GNU GCC сприяє розумінню, як писати ПЗ, що працює на різних системах.
30. Ретельний аналіз кожного кроку лабораторної роботи вчить уважності та точності у програмуванні.
31. Розробка модульного коду дозволяє уникнути дублювання, що полегшує підтримку та оновлення ПЗ.
32. Документування процесу виконання лабораторної роботи покращує комунікацію результатів і зберігає досвід для майбутніх проектів.
33. Розробка тест-драйвера сприяє засвоєнню методики автоматизованого тестування.
34. Практична робота з математичними функціями з бібліотеки <cmath> дає можливість застосовувати теоретичні знання на практиці.
35. Аналіз логів тестування сприяє правильній оцінці якості коду.
36. Розбиття задачі на модулі допомагає зосередитись на вирішенні окремих проблем без зайвих деталей.
37. Створення окремих модулів дозволяє захистити внутрішню логіку від зовнішніх змін.
38. Використання тест-кейсів згідно з міжнародними стандартами підвищує якість і надійність розробленого ПЗ.
39. Реалізація обчислень допомагає закріпити навички роботи з комплексними математичними операціями.
40. Створені модулі та бібліотеки можуть бути використані в інших проектах, що економить час і ресурси.
41. Практика оголошення та ініціалізації змінних допомагає краще розуміти роботу різних типів даних у C++.
42. Аналіз вимог і постановка задач дають можливість оцінити весь цикл розробки ПЗ від планування до тестування.
43. Виконання лабораторної роботи сприяє накопиченню практичного досвіду, який є цінним для майбутніх проектів.
44. Дотримання вимог ДСТУ щодо оформлення звітів та проектної документації формує звички до структурованої роботи.
45. Використання директив #include та інших препроцесорних конструкцій покращує розуміння етапів компіляції.
46. Робота з різними типами літералів допомагає правильно обирати методи їх обробки та виводу.
47. Виділення логічно пов’язаних даних у окремі модулі сприяє кращій організації коду.
48. Застосування модульного підходу є важливим кроком для розробки масштабованих і підтримуваних систем.
49. Аналіз ходу роботи дозволяє об’єктивно оцінити якість реалізованих рішень та процес розробки.
50. Комплексна реалізація поставлених завдань, від проектування до тестування, підтверджує засвоєння ключових технологій і методологій програмування, що є головною метою лабораторної роботи.

