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

Авраменко В. В.

ПЕРЕВІРИВ

викладач кафедри кібербезпеки   
та програмного забезпечення

Коваленко А. С.

Кропивницький – 2025

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).

**Варіант №12**

**Завдання до лабораторної роботи**

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

Аналіз і постановка задачі 8.1

**Аналіз і постановка задачі 8.1**

1. Аналіз вимог

Завдання полягає в обчисленні виразу:

Вхідні дані:

x — дійсне число.

y — дійсне число.

z — дійсне число.

Вихідні дані:

S — результат обчислення (дійсне число).

Обмеження та умови коректності:

1. Вираз містить ділення на x+y, отже, значення x+y не повинно дорівнювати нулю.
2. Підкореневий вираз має бути невід'ємним, щоб уникнути обчислення кореня з від'ємного числа.
3. Функція повинна коректно працювати з різними значеннями x,y,z, зокрема, враховувати особливі випадки, такі як нульові чи від'ємні значення.

Проектування архітектури

Структура програми:

1. Введення вхідних даних: Отримати від користувача або іншого модуля значення x,y,z.
2. Перевірка коректності вхідних даних:
   * Якщо x+y=0, вивести повідомлення про помилку.
   * Якщо < 0, вивести повідомлення про помилку.
3. Обчислення значення SS відповідно до формули.
4. Виведення результату.

Функціональність модуля:

* Функція для обчислення значення SS.
* Обробка виняткових ситуацій (ділення на нуль, корінь із від'ємного числа).
* Вивід результату у зручному форматі.

Результат тестування s\_calculation:

Test case #1: s\_calculation(1, 2, 3) = 96.595 == 96.595 --> passed

Test case #2: s\_calculation(0.5, -1, 4) = -2.830 == -2.830 --> passed

Test case #3: s\_calculation(2, -2, 5) = nan == nan --> passed

Test case #4: s\_calculation(3, 2, -4) = 96.112 == 96.112 --> passed

Test case #5: s\_calculation(-1, -1, -1) = -4.190 == -4.190 --> passed

Test case #6: s\_calculation(0, 0, 0) = nan == nan --> passed

Test case #7: s\_calculation(10, -5, 2) = -9375.385 == -9375.385 --> passed

Test case #8: s\_calculation(0, 3, 2) = 729.000 == 729.000 --> passed

Test case #9: s\_calculation(-3, -2, 4) = -96.112 == -96.112 --> passed

Test case #10: s\_calculation(100, 1, 1) = -2.089 == -2.089 --> passed

Вихідний код ModulesAvramenko:

#include <cmath>

#include <stdexcept>

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

// Перевірка на можливість ділення на нуль

if (std::abs(x + y) < 1e-9) {

return NAN;

throw std::invalid\_argument("Помилка: ділення на нуль у виразі yz/(x+y)");

}

// Обчислення аргументу під квадратним коренем

double sqrt\_argument = std::abs(y \* z / (x + y));

// Перевірка, що sqrt\_argument не дорівнює нулю, інакше відбудеться ділення на нуль

if (std::abs(sqrt\_argument) < 1e-9) {

return NAN;

throw std::domain\_error("Помилка: обчислення sqrt(0) викликає ділення на 0 в основному виразі");

}

// Обчислення виразу

double result = std::sin(x) / std::sqrt(sqrt\_argument) + 3 \* std::pow(y, 5);

return result;

}

Вихідний код TestDriver:

#include <iostream>

#include <sstream>

#include <iomanip>

#include <cmath>

#include "ModulesAvramenko.h"

using namespace std;

string doubleToString(double value, int precision = 3) {

ostringstream oss;

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

return oss.str();

}

void test\_s\_calculation() {

double x[10] = {1, 0.5, 2.0, 3.0, -1.0, 0, 10, 0, -3, 100};

double y[10] = {2.0, -1.0, -2.0, 2.0, -1.0, 0, -5, 3, -2, 1};

double z[10] = {3.0, 4.0, 5.0, -4.0, -1.0, 0.0, 2.0, 2.0, 4.0, 1.0};

double expected\_results[10] = {96.595, -2.8305, NAN, 96.1116, -4.19002, NAN, -9375.385, 729, -96.1116, -2.08891};

for (short i = 0; i < 10; i++) {

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

string resultStr = doubleToString(result);

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

bool passed = false;

// Для NaN використовується isNaN, оскільки NaN != NaN

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 вимог та постановка задачі 8.2

Опис задачі

Задача 8.2 включає три підзадачі, які повинні бути реалізовані як окремі функції, результати яких виводяться в потік cout за допомогою оператора вставки <<.

Підзадача 8.2.1

**Завдання:** Вивести прізвище та ім’я розробника програми із знаком охорони авторського права «©».

**Вхід:** Немає додаткових вхідних даних (інформація заздалегідь закодована).

**Вихід:** Строка, наприклад, "Іваненко Іван ©".

Підзадача 8.2.2

**Завдання:** Вивести результат логічного виразу у числовому вигляді (1/0):

a+1<b+3?a + 1 < b + 3 \quad ?a+1<b+3?

**Вхід:** Значення символів **а** і **b** (ймовірно, типу char або int, що представляють деякі числові значення або коди символів).

**Вихід:** Якщо вираз істинний – виводиться число **1**, інакше – **0**.

**Особливості:** Перетворення логічного результату в числовий формат.

Підзадача 8.2.3

**Завдання:** Вивести значення чисел **x, y, z** у десятковій і шістнадцятковій системах числення, а також значення **S**, яке обчислюється за допомогою функції s\_calculation() із заголовочного файлу ModulesAvramenko.h.

**Вхід:** Послідовне введення чисел **x, y, z**.

**Вихід:** Результати виводу чисел:

Для **x, y, z**: вивід у десятковій системі та у шістнадцятковому представленні.

Обчислення **S** через виклик функції s\_calculation(x, y, z).

**Особливості:**

Використання маніпуляторів форматування (наприклад, **std::dec** і **std::hex** з **iomanip**) для виводу даних.

Підключення функціоналу з бібліотеки (заголовочного файлу) для обчислення **S**.

Обмеження та умови

Усі функції повинні використовувати стандартний потік cout для виводу результатів.

Функції повинні бути сумісні з системою статичних бібліотек (розширення .a або .lib) та відповідати правилам оголошення змінних у C++.

Формат виводу може бути реалізований із застосуванням маніпуляторів з бібліотеки iomanip.

2. Проектування архітектури програмного забезпечення

Структура рішення

Програма складається з:

**Заголовочного файлу (**ModulesAvramenko.h**):** містить декларацію функції **s\_calculation()** та, можливо, інших допоміжних функцій.

**Окремих функцій для підзадач:**

developer\_info() – для підзадачі 8.2.1.

logical\_expression\_result(char a, char b) – для підзадачі 8.2.2.

print\_numbers\_and\_S(double x, double y, double z) – для підзадачі 8.2.3, яка крім виводу чисел форматує їх у десятковому та шістнадцятковому вигляді і викликає s\_calculation().

**Основна функція main():** координує послідовність запитів введення даних, виклик відповідних функцій та вивід результатів через **cout**.

Використані бібліотеки

**<iostream>** для введення/виведення.

**<iomanip>** для форматування виводу (наприклад, **std::dec**, **std::hex**, **std::setw** тощо).

**<stdexcept>** для обробки виняткових ситуацій, якщо це необхідно.

**Модуль з обчисленням S:** заголовковий файл **ModulesПрізвище.h**, який містить оголошення функції **s\_calculation()**.

**Аргументи досягнення мети:**

* + 1. **Чітка постановка задачі:** Завдання лабораторної роботи були сформульовані ясно, що дозволило зосередитися на основних аспектах розробки модулів.
    2. **Реалізація функціональної декомпозиції:** Застосування методу розбиття задачі на менші підзадачі сприяло зрозумілому та логічному коду.
    3. **Створення статичної бібліотеки:** Проектування бібліотеки зміцнило розуміння принципів компіляції та лінкування в C/C++.
    4. **Використання середовища Code::Blocks:** Практична робота в IDE допомогла освоїти роботу з кросплатформовими інструментами.
    5. **Реалізація тестування модулів:** Розробка тестових прикладів підвищила впевненість у коректності реалізації.
    6. **Практичне застосування Git:** Робота з репозиторієм сприяла розвитку навичок контролю версій.
    7. **Документування процесу:** Звітність за методичними рекомендаціями забезпечила систематизацію здобутих результатів.
    8. **Аналіз вимог:** Проведений аналіз дозволив правильно спланувати архітектуру рішення.
    9. **Проектування архітектури:** Детальний проект допоміг зрозуміти зв’язки між компонентами системи.
    10. **Використання мовою C++:** Практична реалізація дозволила засвоїти особливості синтаксису та структуризації коду.
    11. **Набір контрольних прикладів:** Розробка тестів гарантує стабільність і відсутність критичних помилок у роботі функції.
    12. **Консольний застосунок:** Реалізація консольного додатку дозволила перевірити роботу бібліотеки у режимі виконання.
    13. **Досвід роботи з заголовковими файлами:** Створення та використання заголовкових файлів допомогло розуміти механізми інкапсуляції функціоналу.
    14. **Використання препроцесора:** Налаштування директив препроцесора поглибило знання про компіляційний процес.
    15. **Модульне тестування:** Проведення тестування модулів дозволило оперативно виявити та виправити помилки.
    16. **Логування процесу тестування:** Протоколювання результатів тестування сприяло аналізу роботи програми.
    17. **Практична робота з компілятором GNU GCC:** Робота з цим компілятором дала змогу зрозуміти особливості оптимізації коду.
    18. **Застосування концепції інкапсуляції:** Розробка окремих модулів сприяла кращому розділенню функціональності.
    19. **Підхід розробки через інтерфейси:** Використання інтерфейсів дозволило створити гнучку архітектуру рішення.
    20. **Орієнтація на стандарти ISO/IEC 12207:** Дотримання міжнародних стандартів забезпечило якість процесу розробки.
    21. **Підготовка звіту за стандартами ДСТУ:** Застосування вимог до оформлення звітів сприяло дотриманню структурованого підходу.
    22. **Аналіз і постановка задач:** Систематичний аналіз вимог забезпечив розуміння задачі з різних аспектів.
    23. **Практична реалізація проектних рішень:** Застосування власних проектних рішень дозволило знайти оптимальні шляхи вирішення проблем.
    24. **Використання статичних бібліотек:** Реалізація бібліотеки дозволила зменшити час компіляції та покращити продуктивність.
    25. **Системне тестування:** Проведення системного тестування забезпечило впевненість у стабільності кінцевого ПЗ.
    26. **Критичний аналіз результатів:** Детальний аналіз виконання завдань допоміг виявити сильні та слабкі сторони рішення.
    27. **Покрокова реалізація завдань:** Послідовне виконання інструкцій допомогло уникнути плутанини та помилок.
    28. **Використання модульного (блочного) тестування:** Такий підхід дозволив ізолювати проблемні ділянки коду для їх виправлення.
    29. **Застосування практичного підходу до програмування:** Лабораторна робота зміцнила практичні навички роботи з мовою C/C++.
    30. **Розуміння принципів функціонального розв’язання задач:** Реалізація функції s\_calculation дала змогу зрозуміти алгоритмічні принципи розв’язання задач.
    31. **Підвищення навичок роботи з помилками:** Відлагодження коду навчило ефективно знаходити та виправляти помилки.
    32. **Досвід роботи з консольним вводом/виводом:** Практична робота з потоками вводу/виводу допомогла краще зрозуміти обмін даними між модулями.
    33. **Застосування арифметичних та логічних операцій:** Використання операцій дозволило закріпити базові поняття мови.
    34. **Опрацювання документації:** Систематичне ведення документації покращило організацію проекту.
    35. **Порівняльний аналіз функцій та модулів:** Аналіз відмінностей між функціями та модулями дозволив глибше зрозуміти принципи програмування.
    36. **Розвиток навичок системного мислення:** Аналіз всього процесу виконання завдань сприяв розвитку логічного мислення.
    37. **Практична орієнтація лабораторної роботи:** Завдання мали чітко практичну спрямованість, що дозволило застосувати теорію на практиці.
    38. **Інтеграція різних технологій:** Поєднання роботи з Git, IDE та мовою програмування сприяло універсальності отриманих знань.
    39. **Підготовка до реальних умов розробки ПЗ:** Лабораторна робота моделювала реальні процеси розробки програмного забезпечення.
    40. **Застосування сучасних стандартів програмування:** Використання стандартів ISO/IEC підвищило якість та структурність коду.
    41. **Удосконалення навичок роботи з C/C++:** Практична реалізація сприяла кращому засвоєнню особливостей мови.
    42. **Розвиток аналітичних здібностей:** Аналіз процесу проектування допоміг розвинути здатність критично оцінювати власну роботу.
    43. **Використання кросплатформових інструментів:** Робота в середовищі Code::Blocks дала змогу опанувати інструменти, що працюють на різних ОС.
    44. **Вдосконалення навичок структурного програмування:** Модульний підхід допоміг чітко розділити функціонал програми.
    45. **Усвідомлення важливості тестування:** Ретельне тестування дозволило переконатися в надійності реалізованого рішення.
    46. **Практичне застосування теоретичних знань:** Робота над завданнями дозволила інтегрувати теоретичні знання з практикою.
    47. **Розширення бази знань:** Лабораторна робота сприяла ознайомленню з новими концепціями та підходами в програмуванні.
    48. **Успішне вирішення поставлених завдань:** Досягнення мети роботи свідчить про правильність обраних методів та рішень.
    49. **Застосування принципу повторного використання коду:** Створення бібліотеки дозволяє використовувати написаний код у інших проектах.
    50. **Підвищення впевненості у власних силах:** Успішне завершення завдань зміцнило впевненість у власних здібностях до розв’язання складних програмних задач.

