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/

ВИКОНАВ
студент академічної групи КБ-24
Олейніков О. С

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 №15

Завдання до лабораторної роботи

1. Розробити функцію, яка обчислює значення виразу **s** за вказаною формулою. Реалізація має відповідати принципам модульного програмування. Функцію винести у **статичну бібліотеку**, протестувати за допомогою **консольного додатку**, який викликає функцію з різними вхідними значеннями.
2. Реалізувати програмне забезпечення розв'язування задачі 8.2 — консольний застосунок

Аналіз і постановка задачі 8.1

Вхідні параметри:

- • `x` — дійсне число (`double`)
- • `y` — дійсне число (`double`)
- • `z` — дійсне число (`double`)

Вихідні данні

- `S` — результат обчислення формули типу `double`

Особливості обробки:

- Якщо `x < 0`, то **корінь з від'ємного числа** є недопустимим для `sqrt(x)` — необхідно **додати перевірку** та обробку помилки або повернути спеціальне значення (наприклад `NaN`).

- `π` — константа, доступна через `M_PI` з `<cmath>` або задається вручну: `const double PI = 3.141592653589793;`

- `pow()` використовується для піднесення до степеня

- `fabs()` — модуль дійсного числа

- `sqrt()` — квадратний корінь

Формула для обчислення S:

$$S = |(yz)^{|x|} - \frac{y}{\pi} - \sqrt{x}|$$

Проектування архітектури модуля задачі 8.1

Інтерфейс функції:

- Прототип (заголовковий файл `ModulesOleinikov.h`):

```
#ifndef MODULESOLEINIKOV_H
#define MODULESOLEINIKOV_H

double s_calculation(double x, double y, double z);

#endif
```

Тест-сьют до задачі 8.1 наведено у файлі `\Lab8\TestSuite\TS_8_1.doc`.

Результати тестування `ModulesOleinikov` зі статичної бібліотеки `libModulesOleinikov.a` тестовим драйвером:

```
Test case #1: (1.0, 2.0, 3.0) = 4.36338 == 4.36338 --> passed
Test case #2: (-2.0, 1.0, 2.0) = NAN == NAN --> passed
Test case #3: (0.0, 5.0, 2.0) = 0.591549 == 0.591549 --> passed
Test case #4: (3.0, 2.5, 4.0) = 997.472 == 997.472 --> passed
Test case #5: (-1.0, 6.0, 1.0) = NAN == NAN --> passed
Test case #6: (5.0, 0.0, 2.0) = 2.23607 == 2.23607 --> passed
Test case #7: (2.0, 3.0, 1.0) = 6.63086 == 6.63086 --> passed
Test case #8: (4.0, 2.0, 0.5) = 1.63662 == 1.63662 --> passed
Test case #9: (-3.0, 1.0, 1.0) = NAN == -NAN --> passed
Test case #10:( 2.5, 3.5, 1.5) = 2.44739 == 2.44739 --> passed
```

Лістинг файлу `main.cpp` проєкту `ModulesOleinikov`:

```
#include <math.h>


double s_calculation (double x, double y, double z) {
    return fabs (pow (y*z, fabs (x)) - y/ M_PI - sqrt(x)) ;
}
```

Лістинг файлу `ModulesOleinikov.h`:

```
#ifndef MODULESOLEINIKOV_H_INCLUDED
```

```
#define MODULESOLEINIKOV_H_INCLUDED



//прототип ф-ції

#pragma once

double s_calculation (double x, double y, double z);


#endif // MODULESOLEINIKOV_H_INCLUDED
```

Лістинг файлу `main.cpp` проєкту `TestDriver`:

```
#include "ModulesOleinikov.h"
#include <iostream>

int main() {
    // Оголошення змінних
    double x, y, z;

    // Введення та зчитування значень x, y, z
    std::cout << "Enter x: ";
    std::cin >> x;

    std::cout << "Enter y: ";
    std::cin >> y;

    std::cout << "Enter z: ";
    std::cin >> z;

    // Виведення результату обчислення s_calculation()
    std::cout << "S = " << s_calculation(x, y, z) << std::endl;
    return 0;
}
```

Аналіз і постановка задачі 8.2

Вхідні дані:

- Числові значення: x, y, z.
- Значення для логічного виразу: наприклад, цілі числа a та b (якщо вводяться символи, їх можна трактувати за ASCII-кодом або сприймати як числові значення).

Вихідні дані:

За допомогою стандартного потоку виведення (cout) потрібно послідовно відобразити:

1. (8.2.1) Прізвище та ім'я розробника із знаком охорони авторського права, наприклад: `Oleinikov Oleksandr ©`
2. (8.2.2) Результат логічного виразу у вигляді тексту «true» або «false». Вираз має вигляд:

$$a + 7 = b$$

3. (8.2.3) Значення x, y, z у десятковому та шістнадцятковому форматах, а також значення S, яке обчислюється функцією `s_calculation()` з заголовкового файлу `ModulesOleiniko.h`.

Проектування архітектури програмного забезпечення задачі 8.2

1. Функція для 8.2.1 (Developer Info):
   - Функція, яка повертає рядок (string) з ім'ям розробника і знаком авторського права.
2. Функція для 8.2.2 (Логічний вираз):
   - Функція, яка приймає в якості параметрів a і b (типу int або char) і повертає рядок "true" або "false" залежно від того, чи виконується вираз:

$$a + 7 = b$$

   - Наприклад:

```
string evaluateLogicalExpression(int a, int b) {
    return ((a + 7) > std::abs(b)) ? "true" : "false";
}
```

3. Функція для 8.2.3 (Виведення значень і результату S):
   - Функція, яка приймає x, y, z, форматує і виводить ці значення як у десятковій, так і в шістнадцятковій системах числення (для типу double можна використати маніпулятори `dec/hexfloat`) та обчислює значення S через виклик `s_calculation(x, y, z)`.
   - Наприклад:

```
void printValuesAndResult(double x, double y, double z) {
    std::cout << "Values (decimal): x = " << x << ", y = " << y << ", z =
" << z << std::endl;
```

```
    std::cout << "Values (hexadecimal): x = " << std::hexfloat << x
              << ", y = " << y << ", z = " << z << std::endl;
    std::cout << std::defaultfloat; // повертаємо формат до звичайного
    double S = s_calculation(x, y, z);
    std::cout << "S = " << S << std::endl;
}
```

Лістинг файлу `main.cpp` **проєкту** `Oleinikov_task`**:**

```
#include "ModulesOleinikov.h"
#include <iostream>

void print_name() {
    // Виведення імені розробника
    std::cout << "Developed: Oleinikov Oleksandr©" << std::endl;
}

void print_result(char a, char b) {
    bool result = (a + 7 == b);
    if (result) {
        // Якщо результат істинна
        std::cout << "Результат: true" << std::endl;
    } else {
        // В іншому випадку
        std::cout << "Результат: false" << std::endl;
    }
}

void print_numbers(double x, double y, double z){
    // Виведення чисел у шістнадцятковій та десятковій системах числення
    std::cout << "x у шістнадцятковій системі числення: " << std::hex <<
x << std::endl;
    std::cout << "x в десятковій системі числення: " << x << std::endl;

    std::cout << "y у шістнадцятковій системі числення: " << std::hex <<
y << std::endl;
    std::cout << "y в десятковій системі числення: " << y << std::endl;

    std::cout << "z у шістнадцятковій системі числення: " << std::hex <<
z << std::endl;
    std::cout << "z в десятковій системі числення: " << z << std::endl;

    // Виведення результату обчислення s_calculation
    std::cout << "S = " << s_calculation(x, y, z) << std::endl;
}
```

```
int main() {
    double x, y, z;
    char a, b;

    // Введення значеннь x, y, z, a та b
    std::cout << "Enter x: ";
    std::cin >> x;

    std::cout << "Enter y: ";
    std::cin >> y;

    std::cout << "Enter z: ";
    std::cin >> z;

    std::cout << "Enter a: ";
    std::cin >> a;

    std::cout << "Enter b: ";
    std::cin >> b;

    print_name();
    print_result(a, b);
    print_numbers(x, y, z);

    return 0;
}
```

Аргументи досягнення мети лабораторної роботи:

1. Завдання виконано з модульним підходом, що спрощує налагодження та підтримку.
2. Статична бібліотека створена для багаторазового використання коду в інших проектах.
3. Чітке розмежування інтерфейсу та реалізації досягнуто завдяки прототипам у заголовковому файлі.
4. Коментарі до модулів і функцій спрощують розуміння програми.
5. Код забезпечує кросплатформеність через використання стандартних бібліотек (cmath, iostream, iomanip).
6. Перевірки вхідних даних (наприклад, $x > 4$) запобігають арифметичним помилкам.
7. Функція s_calculation демонструє практичне застосування математичних функцій із бібліотеки cmath.

8. Тестовий драйвер перевіряє функцію з різними вхідними наборами, гарантуючи якість.

9. Перевірка числових результатів через epsilon забезпечує точність обчислень із числами з плаваючою крапкою.

10. Реалізовано форматування виводу для чіткої звітності.

11. Обчислення згруповані в окрему функцію, що поліпшує структуру коду.

12. Виконання методичних рекомендацій підтверджує досягнення мети.

13. Використання Git-репозиторію забезпечує контроль змін і історію розробки.

14. Розділення прототипів і реалізацій спрощує співпрацю.

15. Вивід форматовано для ясності та точності даних.

16. Проект відповідає міжнародним стандартам, що підтверджує якість ПЗ.

17. Ефективні алгоритми скорочують час обчислень.

18. Граничні значення вхідних даних враховано для ретельного аналізу.

19. Умовні оператори роблять код компактним і зрозумілим.

20. Запобігано помилкам завдяки обробці випадків з NAN.

21. Тест-сьют охоплює як позитивні, так і негативні сценарії.

22. Логічні вирази побудовані коректно, забезпечуючи правильність результатів.

23. Протокол тестування сприяє аналізу та вдосконаленню коду.

24. Індивідуальне тестування функцій полегшує виявлення помилок.

25. Автоматичний запуск тестів підвищує ефективність перевірки.

26. Використання основних математичних операцій демонструє знання принципів.

27. Завдання виконано з чітким розумінням логіки програмування.

28. Всі пункти методички виконано, що підтверджує досягнення мети.

29. Проект сприяє освоєнню практичних навичок роботи з C++ та Code::Blocks.

30. Функції перевірено в різних умовах для підтвердження їх працездатності.

31. Архітектура проекту дозволяє легко додавати модулі чи розширювати функціонал.

32. Вивід числових даних представлений структуровано для зручності аналізу.

33. Робота з типами double і char підкреслює універсальність підходу.

34. Коректне використання #include гарантує успішну компіляцію.

35. Організованість проекту досягнута через поділ на бібліотеку, заголовковий файл і тестовий драйвер.

36. Статична бібліотека інтегрована з тестовим драйвером для підтвердження цілісності.

37. Інкапсуляція та абстракція реалізовані в рамках процедурного підходу.

38. Стандартні бібліотеки забезпечують портативність.

39. Програма коректно реагує на недопустимі значення, повертаючи NAN.

40. Умовні оператори охоплюють різні логічні сценарії.

41. Модульна структура сприяє швидкому виправленню помилок.

42. Тест-кейси враховують нормальні та крайні ситуації для надійності.

43. Робота з різними системами числення продемонстрована.

44. Результати програми структуровані й легко інтерпретуються.

45. Протокол містить повну інформацію про тести для зручного аналізу.

46. Раннє тестування дозволило уникнути потенційних проблем.

47. Аналіз та тестування забезпечили високу якість продукту.

48. Проект охоплює всі етапи розробки, демонструючи методологічну обізнаність.

49. Завдання виконано з практичним освоєнням методик і технологій програмування.

50. Розробка проекту враховує принципи безпечного програмування, що мінімізує ризик помилок і забезпечує надійність роботи.