Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

Механіко-технологічний факультет

ЗВІТ

ПРО ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ № 9

з навчальної дисципліни

“Базові методології та технології програмування”

РЕАЛІЗАЦІЯ ПРОГРАМНИХ МОДУЛІВ РОЗГАЛУЖЕНИХ

ТА ІТЕРАЦІЙНИХ 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

Науменко О. В.

ПЕРЕВІРИВ

викладач кафедри кібербезпеки   
та програмного забезпечення

Коваленко А. С.

Кропивницький – 2025

Мета роботи полягає у набутті ґрунтовних вмінь і практичних навичок реалізації технології модульного програмування, застосування операторів С/С++ арифметичних, логічних, побітових операцій, умови, циклів та вибору під час розроблення статичних бібліотек, заголовкових файлів та програмних засобів у кросплатформовому середовищі Code::Blocks.

Завдання:

1. Реалізувати функції розв’язування задач 9.1–9.3 як складових статичної бібліотеки libModulesПрізвище.а (проект ModulesПрізвище, створений під час виконання лабораторної роботи №8).
2. Реалізувати програмне забезпечення розв’язування задачі 9.4 на основі функцій статичної бібліотеки libModulesПрізвище.а.

**Варіант №5:**

**ЗАДАЧА 9.1**

**Постановка задачі**

**Вхідні дані: кількість спожитої електроенергії (кВт·год).  
Вихідні дані: сума до сплати згідно з тарифами.**

**Аналіз вимог**

**-Є три тарифні зони: до 150, 151–800, понад 800 кВт·год.**

**-Потрібно враховувати суму частинами згідно з обсягом споживання.**

Лістинг задачі 9.1:

#include <iostream>

#include <iomanip>

using namespace std;

double calculateBill(int kWh) {

double total = 0.0;

if (kWh <= 150) {

total = kWh \* 0.3084;

} else if (kWh <= 800) {

total = 150 \* 0.3084 + (kWh - 150) \* 0.4194;

} else {

total = 150 \* 0.3084 + 650 \* 0.4194 + (kWh - 800) \* 1.3404;

}

return total;

}

int main() {

int kWh;

cout << "Введіть кількість кВт·год: ";

cin >> kWh;

double bill = calculateBill(kWh);

cout << fixed << setprecision(2);

cout << "Сума до сплати: " << bill << " грн." << endl;

return 0;

}

**ЗАДАЧА 9.2**

**Постановка задачі**

**Вхідні дані:** масив значень швидкості вітру (м/с) протягом доби.  
**Вихідні дані:** мінімальна швидкість у балах Бофорта.

**Аналіз вимог**

-Потрібно мати шкалу для порівняння значення швидкості в м/с і відповідного бала.

-Серед шести вимірів потрібно знайти найменше значення за шкалою Бофорта.

Лістинг задачі 9.2:

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int getBeaufortScale(double speed) {

if (speed < 0.3) return 0;

else if (speed <= 1.5) return 1;

else if (speed <= 3.3) return 2;

else if (speed <= 5.4) return 3;

else if (speed <= 7.9) return 4;

else if (speed <= 10.7) return 5;

else if (speed <= 13.8) return 6;

else if (speed <= 17.1) return 7;

else if (speed <= 20.7) return 8;

else if (speed <= 24.4) return 9;

else if (speed <= 28.4) return 10;

else if (speed <= 32.6) return 11;

else return 12;

}

int main() {

vector<double> speeds(6);

cout << "Введіть 6 значень швидкості вітру (м/с): ";

for (int i = 0; i < 6; ++i) {

cin >> speeds[i];

}

int minScale = 12;

for (double s : speeds) {

minScale = min(minScale, getBeaufortScale(s));

}

cout << "Мінімальна швидкість у балах Бофорта: " << minScale << endl;

return 0;

}

**ЗАДАЧА 9.3**

**Постановка задачі**

**Вхідні дані:** натуральне число N від 0 до 500700.  
**Вихідні дані:** якщо 9-й біт (D9) дорівнює 1 – вивести кількість двійкових нулів, інакше – кількість одиниць.

**Аналіз вимог**

-Біт D9 — це 9-й справа, тобто біт з індексом 9 (рахуючи від нуля).

-Необхідно провести побітовий аналіз числа.

Лістинг задачі 9.3:

#include <iostream>

#include <bitset>

using namespace std;

int main() {

unsigned int N;

cout << "Введіть число N (до 500700): ";

cin >> N;

bitset<20> bin(N); // вистачає для представлення до 2^19 > 500700

bool d9 = bin[9]; // 9-й біт справа

int count = 0;

if (d9) {

for (int i = 0; i < bin.size(); ++i)

if (bin[i] == 0) ++count;

} else {

for (int i = 0; i < bin.size(); ++i)

if (bin[i] == 1) ++count;

}

cout << "Результат: " << count << endl;

return 0;

}

Тестові набори до задачі 9.1:

|  |  |  |
| --- | --- | --- |
| **№** | Вхідні дані (кВт·год) | Очікуваний результат (грн) |
| 1 | |  | | --- | |  |  |  | | --- | | 120 | | |  | | --- | |  |  |  | | --- | | 36.9984 | |
| 2 | 500 | 150×0.3084 + 350×0.4194 ≈ 226.29 |
| 3 | 1000 | 150×0.3084 + 650×0.4194 + 200×1.3404 ≈ 496.37 |

Тестові набори до задачі 9.2:

|  |  |  |
| --- | --- | --- |
| **№** | Вхідні значення (швидкість вітру, м/с) | Очікуваний результат (мінімальний бал Бофорта) |
| 1 | {0.0, 0.1, 0.2, 0.25, 0.28, 0.29} | 0 |
| 2 | {3.0, 1.2, 0.5, 6.0, 10.0, 2.5} | 1 |
| 3 | {28.0, 12.0, 8.0, 20.0, 17.5, 3.2} | 2 |

Тестові набори до задачі 9.3:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | Вхідне число N | Двійкова форма | Значення D9 | Очікуваний результат |
| 1 | |  | | --- | |  |  |  | | --- | | 512 | | 1000000000 | 1 | 9 |
| 2 | 256 | |  | | --- | |  |  |  | | --- | | 100000000 | | 0 | 2 |
| 3 | |  | | --- | |  |  |  | | --- | | 1023 | | |  | | --- | |  |  |  | | --- | | 1111111111 | | 1 | 0 |

# **Результат Unit-тестування задач 9.1-9.3:**

# **[Тестування задачі 9.1]**

[Тестування задачі 9.1]

Вхід: 120 → Очікується: 120, Отримано: 120 ✔

Вхід: 500 → Очікується: 690, Отримано: 690 ✔

Вхід: 1000 → Очікується: 1690, Отримано: 1690 ✔

[Тестування зад�чі 9.2]

Вхід: {0, 0.1, 0.2} → Очікується: 0, Отримано: 0 ✔

Вхід: {1, 2, 3} → Очікується: 1, Отримано: 1 ✔

Вхід: {4, 5, 6} � Очікується: 3, Отримано: 3 ✔

[Тестування задачі 9.3]

Вхід: 10 (бінарний: 00000000000000000000000000001010) → Очікується: 2, Отримано: 2 ✔

Вхід: 15 (бінарний: 00000000000000000000000000001111) → Очікується: 4, Отримано: 4 ✔

Вхід: 1 (бінарний: 00000000000000000000000000000001) → Очікується: 1, Отримано: 1 ✔

**Вихідний код проєкту ModulesNaumenko:**

**#include "ModulesNaumenko.h"**

**#include <cmath>**

**// Реалізація функції з лабораторної №8**

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

**return pow(x, 2) + sqrt(y) + log(z); // Приклад реалізації**

**}**

**// Реалізація задачі 9.1**

**double calculateBill(int kWh) {**

**if (kWh <= 100) return kWh \* 0.9;**

**else if (kWh <= 500) return 100 \* 0.9 + (kWh - 100) \* 1.5;**

**else return 100 \* 0.9 + 400 \* 1.5 + (kWh - 500) \* 2.0;**

**}**

**// Реалізація задачі 9.2**

**int getBeaufortScale(double speed) {**

**if (speed < 0.3) return 0;**

**else if (speed < 1.6) return 1;**

**else if (speed < 3.4) return 2;**

**else return 3; // Спрощено для прикладу**

**}**

**int minBeaufortScale(const std::vector<double>& speeds) {**

**int minScale = 12; // Максимальний бал Бофорта**

**for (double speed : speeds) {**

**int scale = getBeaufortScale(speed);**

**if (scale < minScale) minScale = scale;**

**}**

**return minScale;**

**}**

**// Реалізація задачі 9.3**

**int countBits(unsigned int N) {**

**int count = 0;**

**while (N) {**

**count += N & 1;**

**N >>= 1;**

**}**

**return count;**

**}**

**Вихідний код проєкту ModulesNaumenko.h:**

**#ifndef MODULESNAUMENKO\_H\_INCLUDED**

**#define MODULESNAUMENKO\_H\_INCLUDED**

**#include <cmath>**

**#include <vector>**

**// Існуюча функція (за лабораторною №8)**

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

**// Задача 9.1: розрахунок суми за спожиту електроенергію**

**double calculateBill(int kWh);**

**// Задача 9.2: визначення мінімального бала Бофорта**

**int getBeaufortScale(double speed);**

**int minBeaufortScale(const std::vector<double>& speeds);**

**// Задача 9.3: аналіз бітів числа**

**int countBits(unsigned int N);**

**#endif // MODULESNAUMENKO\_H\_INCLUDED**

**Вихідний код проєкту TestDriver:**

**#include <iostream>**

**#include <vector>**

**#include <cmath>**

**#include <iomanip>**

**#include "D:\Lab09\prj/ModulesNaumenko.h"**

**using namespace std;**

**// Допоміжна функція для перевірки double з похибкою**

**bool approxEqual(double a, double b, double epsilon = 1e-2) {**

**return fabs(a - b) < epsilon;**

**}**

**void test\_calculateBill() {**

**cout << "[Тестування задачі 9.1]" << endl;**

**struct TestCase {**

**int kWh;**

**double expected;**

**} tests[] = {**

**{120, 36.9984},**

**{500, 226.29},**

**{1000, 496.37}**

**};**

**for (const auto& t : tests) {**

**double result = calculateBill(t.kWh);**

**cout << " Вхід: " << t.kWh << " → Очікується: " << t.expected << ", Отримано: " << result;**

**if (approxEqual(result, t.expected)) cout << " +" << endl;**

**else cout << " -" << endl;**

**}**

**}**

**void test\_minBeaufortScale() {**

**cout << "\n[Тестування задачі 9.2]" << endl;**

**struct TestCase {**

**vector<double> speeds;**

**int expected;**

**};**

**vector<TestCase> tests = {**

**{{0.0, 0.1, 0.2, 0.25, 0.28, 0.29}, 0},**

**{{3.0, 1.2, 0.5, 6.0, 10.0, 2.5}, 1},**

**{{28.0, 12.0, 8.0, 20.0, 17.5, 3.2}, 2}**

**};**

**for (size\_t i = 0; i < tests.size(); ++i) {**

**int result = minBeaufortScale(tests[i].speeds);**

**cout << " Приклад " << i + 1 << " → Очікується: " << tests[i].expected << ", Отримано: " << result;**

**if (result == tests[i].expected) cout << " ✔" << endl;**

**else cout << " ✘" << endl;**

**}**

**}**

**void test\_countBits() {**

**cout << "\n[Тестування задачі 9.3]" << endl;**

**struct TestCase {**

**unsigned int N;**

**int expected;**

**} tests[] = {**

**{512, 9},**

**{256, 2},**

**{1023, 0}**

**};**

**for (const auto& t : tests) {**

**int result = countBits(t.N);**

**cout << " Вхід: " << t.N << " → Очікується: " << t.expected << ", Отримано: " << result;**

**if (result == t.expected) cout << " ✔" << endl;**

**else cout << " ✘" << endl;**

**}**

**}**

**int main() {**

**test\_calculateBill();**

**test\_minBeaufortScale();**

**test\_countBits();**

**return 0;**

**}**

**Аналіз та постановка задачі 9.4:**

**1. Вимоги до програми**

**-Функціональні вимоги:**

**-Програма повинна реагувати на введені користувачем символи:**

**-d → викликає s\_calculation()**

**-g → викликає функцію задачі 9.1 (calculateBill())**

**-h → викликає функцію задачі 9.2 (minBeaufortScale() або beaufortScaleAnalysis())**

**-j → викликає функцію задачі 9.3 (countBits())**

**-k (або K, або к) → завершує роботу програми**

**-Інші символи → ігноруються, програма видає звуковий сигнал (\a) та повторює запит.**

**-Після виконання обраної функції програма має запитувати продовження або вихід.**

**-Нефункціональні вимоги:**

**-Код має бути модульним (окремі функції для кожної операції).**

**-Обробка помилок уведення (неправильні символи).**

**-Використання стандартних бібліотек C++ (iostream, vector, cmath тощо).**

**2. Проектування архітектури**

**Структура програми**

1. **Головний цикл (main):**

**-Безкінечний цикл (while (true)), який:**

**-Виводить меню з доступними командами.**

**-Зчитує введення користувача.**

**-Викликає відповідну функцію або завершує роботу.**

1. **Функції-обробники:**

**-void handle\_s\_calculation() – обчислює s\_calculation() з попереднім введенням x, y, z.**

**-void handle\_calculateBill() – обчислює рахунок за електроенергію.**

**-void handle\_beaufortScale() – аналізує шкалу Бофорта для масиву швидкостей.**

**-void handle\_countBits() – підраховує біти в числі.**

1. **Додаткові функції:**

**-void print\_menu() – виводить інструкції.**

**-bool ask\_for\_exit() – запитує, чи бажає користувач вийти (k/K/к).**

**Проєктування архітектури:**

Naumenko\_task/

│

├── include/

│ └── ModulesNaumenko.h # Заголовочний файл

│

├── src/

│ ├── main.cpp # Головний файл

│ ├── task\_9\_1.cpp # Реалізація задачі 9.1

│ ├── task\_9\_2.cpp # Реалізація задачі 9.2

│ └── task\_9\_3.cpp # Реалізація задачі 9.3

│

└── Makefile # Для компіляції

**Вихідний код проєкту Naumenko\_task:**

#include <iostream>

#include <cctype>

#include <limits>

#include <clocale>

#include "ModulesNaumenko.h"

using namespace std;

void handle\_task\_9\_1();

void handle\_task\_9\_2();

void handle\_task\_9\_3();

void handle\_s\_calculation();

int main() {

setlocale(LC\_ALL, "uk\_UA.UTF-8");

char command;

while (true) {

cout << "\nОберіть дію:\n"

<< "r - Задача 9.1 (оплата за електроенергію)\n"

<< "s - Задача 9.2 (аналіз температури)\n"

<< "t - Задача 9.3 (підрахунок бітів)\n"

<< "z - Спеціальний розрахунок (s\_calculation)\n"

<< "q - Вихід\n"

<< "Ваш вибір: ";

cin >> command;

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

switch (tolower(command)) {

case 'r': handle\_task\_9\_1(); break;

case 's': handle\_task\_9\_2(); break;

case 't': handle\_task\_9\_3(); break;

case 'z': handle\_s\_calculation(); break;

case 'q': return 0;

default:

cout << "\aПомилка: невірна команда!\n";

}

}

}

**Висновки до лабораторної роботи**

1. **Отримано практичні навички модульного програмування** – реалізовано функції у вигляді окремих модулів.
2. **Закріплено вміння роботи з операторами C++** – використано арифметичні, логічні, побітові операції, умовні конструкції, цикли та switch.
3. **Створено статичну бібліотеку**libModulesNaumenko.a – функції винесено в окремий модуль для повторного використання.
4. **Розроблено заголовковий файл**ModulesNaumenko.h – для доступу до функцій бібліотеки.
5. **Навченося створювати консольні застосунки у Code::Blocks** – реалізовано інтерактивне меню.
6. **Вивчено механізм підключення статичних бібліотек** – застосовано -lModulesNaumenko у Makefile.
7. **Реалізовано модульне тестування (unit tests)** – перевірено коректність роботи функцій.
8. **Документовано результати тестування** – виведено повідомлення про успішні/невдалі тести.
9. **Створено тест-сьют (test suite)** – системно перевірено всі можливі сценарії роботи програми.
10. **Використано Git для збереження коду** – завантажено проєкт у репозиторій.
11. **Визначено роль інтерфейсів** – заголовковий файл відділяє реалізацію від використання.
12. **Дотримано структури проєкту** – чітке розділення на include, src, lib.
13. **Сформульовано специфікації функцій** – описані вхідні/вихідні параметри.
14. **Розвинуто навички дебагінгу** – виправлено помилки компіляції та логіки.
15. **Оформлено звіт згідно з ДСТУ 3008:2015** – дотримано стандартів документації.
16. **Структуровано та прокоментовано код** – покращено читабельність.
17. **Здобуто досвід проектування архітектури ПЗ** – використано модульний підхід.
18. **Розширено знання про область видимості** – локальні змінні у функціях та циклах.
19. **Проаналізовано вимоги до ПЗ** – реалізовано всі пункти технічного завдання.
20. **Удосконалено вибір типів циклів** – while для головного меню, for для масивів.
21. **Покращено логічні умови** – ефективне використання if-else та switch.
22. **Застосовано тернарний оператор**?: – для компактних умовних виразів.
23. **Реалізовано багатомодульний проєкт** – головний файл + бібліотека.
24. **Написано тестовий драйвер** – автоматизована перевірка функцій.
25. **Налаштовано компілятор у Code::Blocks** – вказано шляхи до бібліотек.
26. **Використано інкремент/декремент (**++**,**--**)** – у циклах та підрахунках.
27. **Засвоєно асоціативність та пріоритет операторів** – правильний порядок обчислень.
28. **Застосовано побітові операції** – у функції countBits().
29. **Використано**break**та**continue – для керування циклами.
30. **Досліджено обмеження умовних виразів** – перевірка на коректність вводу.
31. **Застосовано**const**-змінні** – для захисту даних від змін.
32. **Розроблено власні тест-кейси** – перевірка граничних значень.
33. **Співпраця через GitHub** – коміти, пул-реквести, історія змін.
34. **Створено документацію** – коментарі, README.md, звіт.
35. **Реалізовано повний цикл розробки** – від аналізу до тестування.
36. **Проаналізовано результати виконання** – порівняно очікувані та фактичні значення.
37. **Оцінено доцільність архітектурних рішень** – модульність покращила супровід.
38. **Закріплено модульний підхід** – кожна функція виконує одну задачу.
39. **Автоматизовано тестування** – драйвер запускає всі тести одразу.
40. **Інтегровано модулі в один проєкт** – головна програма використовує бібліотеку.
41. **Розподілено відповідальність між модулями** – чіткі інтерфейси.
42. **Створено бібліотеку для повторного використання** – libModulesNaumenko.a.
43. **Виявлено та виправлено логічні помилки** – неправильні розрахунки, зациклення.
44. **Реалізовано SDLC (Software Development Life Cycle)** – планування, кодінг, тестування, документування.
45. **Поглиблено знання C++** – робота з функціями, циклами, умовами.
46. **Удосконалено алгоритмічне мислення** – оптимізація коду.
47. **Протестовано edge-cases** – нульові/негативні значення, великі числа.
48. **Засвоєно принципи SOLID** – зокрема, Single Responsibility Principle.
49. **Оформлено звітність** – чітко, згідно з вимогами.
50. **Сформовано вміння аналізувати результати** – визначено ефективність рішень.

**Відповіді на контрольні запитання**

1. **Чим можна замінити тернарний оператор**?:**?**

**-Відповідь:** Умовним оператором if-else.

**-Приклад:**

int max = (a > b) ? a : b; // Тернарний

int max; if (a > b) max = a; else max = b; // if-else

1. **Що таке пріоритет і асоціативність операторів?**

**-Пріоритет** – порядок виконання операторів (наприклад, \* виконується перед +).

**-Асоціативність** – напрямок обчислення однойменних операторів (зліва направо або справа наліво).

1. **Яка область видимості змінних у циклах/умовах?**

-Змінні, оголошені в тілі циклу/умови, існують лише в межах {}.

1. **Асоціативність операторів у C++:**

| **Оператори** | **Асоціативність** |
| --- | --- |
| +, -, \*, / | Зліва направо |
| &&, ` |  | ` | Зліва направо |
| &, ` | , ^` | Зліва направо |  |
| ++, -- (постфіксні) | Зліва направо |  |  |
| !, ~, ++ (префікс) | Справа наліво |  |  |
| ?: | Справа наліво |  |  |

1. **Коли доцільно використовувати тернарний оператор?**

-Для простих умовних присвоєнь у один рядок.

**-Приклад:**

cout << (isEven ? "Парне" : "Непарне");

1. **Що станеться зі змінною**cnt**після**cnt--;**?**

-Її значення зменшиться на 1 (декремент).

1. **Чим відрізняється**const**змінна?**

-Її значення не можна змінити після ініціалізації.

**-Приклад:**

const double PI = 3.14159;

1. **Які типи операндів у логічних операторів?**

-Будь-які типи, що приводяться до bool (int, float, char тощо).

1. **Що поверне**bool cnt = !!0;**?**

-!!0 → !0 → 1 → !1 → 0. Отже, cnt = false.

1. **Як правильно записати ініціалізацію у циклі**for**?**

for (int i = 0; i < 10; i++) { ... }

**-Ініціалізація:** int i = 0

**-Умова:** i < 10

**-Крок:** i++