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

Марченко К. О.

ПЕРЕВІРИВ

викладач кафедри кібербезпеки   
та програмного забезпечення

Коваленко А. С.

Кропивницький – 2025

**Тема**: Реалізація програмних модулів розгалужених та ітераційних обчислювальних процесів

**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.

Варіант №20

**Завдання до лабораторної роботи**:

1. Реалізувати функції розв’язування задач 9.1–9.3 як складових статичної бібліотеки libModulesMarchenko.а (проект ModulesMarchenko, створений під час виконання лабораторної роботи №8).

2. Реалізувати програмне забезпечення розв’язування задачі 9.4 на основі функцій статичної бібліотеки libModulesMarchenko.а.

# ЗАДАЧА 9.1

**Формулювання задачі**

За умовами завдання, згідно з трудовим договором, працівник щодня отримує заробітну плату за робочий день, що складається з 8 годин, при погодинній оплаті **450 грн/год**. Оплата за робочий період визначається на підставі відпрацьованих днів за 2 тижні (всього 10 робочих змін за стандартний робочий тиждень).  
**Вхідні дані:**

* Кількість відпрацьованих днів за звітний період (2 тижні).

**Вихідні дані:**

1. **Нарахована сума зарплати (грн):** Розраховується як кількість відпрацьованих днів × 8 годин/день × 450 грн/год.
2. **Розмір вирахуваних податків (грн):** Потрібно врахувати:
   * 15% податку на прибуток фізичної особи;
   * 2% до Пенсійного фонду;
   * 0,6% до Фонду загальнообов’язкового державного соціального страхування на випадок безробіття;
   * 1% загальнообов’язкового державного соціального страхування у зв’язку з тимчасовою втратою працездатності.

Розрахунок кожного податку виконується від нарахованої суми зарплати.

1. **Сума до виплати (грн):** Обчислюється як нарахована зарплата мінус сумарні податкові відрахування.

**Аналіз задачі та постановка алгоритму**

1. **Розрахунок нарахованої зарплати:**
   * Визначити добову норму зарплати: Добова зарплата= 8 × 450 грн.
   * Помножити добову зарплату на кількість відпрацьованих днів.
2. **Розрахунок податків:**

* Обчислити суму податку для кожного відрахування:  
  Податокi = Нарахована зарплата×Відсотокi
* Загальна сума податків – сума всіх окремих податків.

1. **Розрахунок суми до виплати:**

* Вирахувати суму податків із нарахованої зарплати:

Виплата = Нарахована зарплата − Сума податків.

# ЗАДАЧА 9.2

**Формулювання задачі**

Завдання полягає у конвертації розміру чоловічих шкарпеток, заданого в українській системі, у відповідники розмірів за системами США та ЄС.

**Вхідні дані:** Розмір шкарпеток за українською системою (ціле число).

**Вихідні дані:** Відповідний розмір для США та ЄС.

**Аналіз задачі та постановка алгоритму**

1. **Визначення таблиці відповідності:** На основі поданої таблиці маємо такі відповідності:
   * Україна → США/Великобританія та ЄС:
     + 23 → 8 (США), 37/38 (ЄС);
     + 25 → 9 (США), 39/40 (ЄС);
     + 27 → 10 (США), 41/42 (ЄС);
     + 29 → 11 (США), 43/44 (ЄС);
     + 31 → 12 (США), 45/46 (ЄС).
2. **Логіка пошуку відповідностей:**
   * Можна використати конструкцію умов (if-else або switch-case) для вибору відповідного набору.
   * Рекомендовано реалізувати функцію, яка приймає розмір (Україна) і повертає відповідники для США та ЄС.
   * При бажанні, можна також застосувати тернарний оператор для вибору варіантів, хоча вимога про тернарний оператор конкретно зазначена в задачі 9.3.

# ЗАДАЧА 9.3

**Формулювання задачі**

Завдання полягає у підрахунку бітів у двійковому представленні числа N залежно від значення молодшого біту D0.

**Вхідні дані:**

* Натуральне число N (діапазон від 0 до 7,483,650).

**Вихідні дані:**

* Якщо біт D0 (молодший біт числа) дорівнює 0, потрібно вивести кількість нулів у двійковому представленні числа N.
* Якщо ж біт D0 дорівнює 1, потрібно вивести кількість одиниць у двійковому представленні числа N.

**Примітка:** Рекомендується використати тернарний оператор «?:» під час підрахунку бінарних символів.

**Аналіз задачі та постановка алгоритму**

1. **Отримання двійкового представлення числа:**
   * Можна використати стандартні алгоритми для перетворення числа у рядок бітів або безпосередньо аналізувати біти шляхом операцій зсувів.
2. **Перевірка молодшого біту D0:**

* Для цього застосовують побітову операцію AND з 1:
  + Якщо N & 1 == 0 – молодший біт рівний 0.
  + Інакше – рівний 1.

1. **Підрахунок кількості бітів:**
   * Якщо D0 дорівнює 0, пройти по кожному біту двійкового рядка і підрахувати символи '0'.
   * Якщо D0 дорівнює 1, підрахувати символи '1'.
   * Рекомендується використати тернарний оператор для вибору між підрахунком нулів та одиниць.

# Тестові варіанти до Задачі 9.1

1. **Тест-кейс 1:**
   * **Вхід:** кількість відпрацьованих днів = 0
   * **Обчислення:**
     + Зарплата = 0 × 8 × 450 = 0 грн
     + Податки = 0 грн
     + Сума до виплати = 0 грн
   * **Очікуваний результат:**
     + Зарплата: 0 грн
     + Податки: 0 грн
     + Сума до виплати: 0 грн
2. **Тест-кейс 2:**
   * **Вхід:** кількість відпрацьованих днів = 5
   * **Обчислення:**
     + Зарплата = 5 × 8 × 450 = 18 000 грн
     + Податок на прибуток = 18 000 × 0.15 = 2 700 грн
     + Податок до Пенсійного фонду = 18 000 × 0.02 = 360 грн
     + Податок на безробіття = 18 000 × 0.006 = 108 грн
     + Податок при втраті працездатності = 18 000 × 0.01 = 180 грн
     + Сумарні податки = 2 700 + 360 + 108 + 180 = 3 348 грн
     + Сума до виплати = 18 000 − 3 348 = 14 652 грн
   * **Очікуваний результат:**
     + Зарплата: 18 000 грн
     + Податки: 3 348 грн
     + Сума до виплати: 14 652 грн
3. **Тест-кейс 3:**
   * **Вхід:** кількість відпрацьованих днів = 10 (максимум для двотижневого періоду)
   * **Обчислення:**
     + Зарплата = 10 × 8 × 450 = 36 000 грн
     + Податок на прибуток = 36 000 × 0.15 = 5 400 грн
     + Податок до Пенсійного фонду = 36 000 × 0.02 = 720 грн
     + Податок на безробіття = 36 000 × 0.006 = 216 грн
     + Податок при втраті працездатності = 36 000 × 0.01 = 360 грн
     + Сумарні податки = 5 400 + 720 + 216 + 360 = 6 696 грн
     + Сума до виплати = 36 000 − 6 696 = 29 304 грн
   * **Очікуваний результат:**
     + Зарплата: 36 000 грн
     + Податки: 6 696 грн
     + Сума до виплати: 29 304 грн

# Тестові варіанти до Задачі 9.2

1. **Тест-кейс 1:**
   * **Вхід:** розмір шкарпеток (UA) = 23
   * **Очікуваний результат:**
     + Розмір США: 8
     + Розмір ЄС: "37/38"
2. **Тест-кейс 2:**
   * **Вхід:** розмір шкарпеток (UA) = 27
   * **Очікуваний результат:**
     + Розмір США: 10
     + Розмір ЄС: "41/42"
3. **Тест-кейс 3:**
   * **Вхід:** розмір шкарпеток (UA) = 31
   * **Очікуваний результат:**
     + Розмір США: 12
     + Розмір ЄС: "45/46"

*Примітка:* Якщо вхідне значення не відповідає жодному з визначених розмірів (наприклад, 24, 26 тощо), необхідно реалізувати обробку помилок або повернення повідомлення про некоректний ввід.

# Тестові варіанти до Задачі 9.3

**Алгоритм задачі:**

1. Перетворити число N у двійкове представлення.
2. Перевірити молодший біт D0 (за допомогою операції N&1):
   * Якщо D0 = 0, рахувати кількість нулів у двійковому представленні.
   * Якщо D0 = 1, рахувати кількість одиниць у двійковому представленні.
3. Для підрахунку рекомендовано використовувати тернарний оператор.

**Набір тестів:**

1. **Тест-кейс 1:**
   * **Вхід:** N=0N = 0
   * **Обчислення:**
     + Двійкове представлення: "0"
     + Молодший біт D0 = 0
     + Кількість нулів: 1 (один символ "0")
   * **Очікуваний результат:** 1
2. **Тест-кейс 2:**
   * **Вхід:** N=6
   * **Обчислення:**
     + Двійкове представлення: "110"
     + Молодший біт D0 = 0 (остання цифра 0)
     + Підрахунок: символи "1", "1", "0" → нульових символів: 1
   * **Очікуваний результат:** 1
3. **Тест-кейс 3:**
   * **Вхід:** N=7
   * **Обчислення:**
     + Двійкове представлення: "111"
     + Молодший біт D0 = 1
     + Підрахунок: усі символи "1" → одиниць: 3
   * **Очікуваний результат:** 3

# Результати Unit-тестування:

=== Тестовий драйвер для модульного тестування задач 9.1, 9.2 та 9.3 ===

=== Тестування задачі 9.1: Зарплата та податки ===

[TC1] Вхід: daysWorked = 0

Отримано: grossSalary = 0, totalTaxes = 0, netSalary = 0

Очікувано: 0, 0, 0 -> PASSED

[TC2] Вхід: daysWorked = 5

Отримано: grossSalary = 18000, totalTaxes = 3348, netSalary = 14652

Очікувано: 18000, 3348, 14652 -> PASSED

[TC3] Вхід: daysWorked = 10

Отримано: grossSalary = 36000, totalTaxes = 6696, netSalary = 29304

Очікувано: 36000, 6696, 29304 -> PASSED

Результат тестування задачі 9.1: ВСІ ТЕСТИ PASSED

=== Тестування задачі 9.2: Конвертація розміру шкарпеток ===

[TC1] Вхід: uaSize = 23

Отримано: US = 8, EU = 37/38

Очікувано: US = 8, EU = 37/38 -> PASSED

[TC2] Вхід: uaSize = 27

Отримано: US = 10, EU = 41/42

Очікувано: US = 10, EU = 41/42 -> PASSED

[TC3] Вхід: uaSize = 31

Отримано: US = 12, EU = 45/46

Очікувано: US = 12, EU = 45/46 -> PASSED

Результат тестування задачі 9.2: ВСІ ТЕСТИ PASSED

=== Тестування задачі 9.3: Підрахунок бінарних символів ===

[TC1] Вхід: N = 0

Отримано: 1

Очікувано: 1 -> PASSED

[TC2] Вхід: N = 6

Отримано: 1

Очікувано: 1 -> PASSED

[TC3] Вхід: N = 7

Отримано: 3

Очікувано: 3 -> PASSED

Результат тестування задачі 9.3: ВСІ ТЕСТИ PASSED

# Вихідний код ModulesMarchenko:

#include <cmath>

#include <stdexcept>

#include <iostream>

#include <string>

#include <bitset>

#include <algorithm>

#include "ModulesMarchenko.h"

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;

}

// Функція calculateSalary розраховує нараховану суму зарплати, податкові відрахування та чисту суму до виплати

SalaryInfo calculateSalary(int daysWorked) {

const int HOURS\_PER\_DAY = 8;

const int RATE\_PER\_HOUR = 450;

// Обчислення загальної нарахованої зарплати

double grossSalary = daysWorked \* HOURS\_PER\_DAY \* RATE\_PER\_HOUR;

// Розрахунок відрахувань:

// 15% податку на прибуток фізособи, 2% до Пенсійного фонду, 0,6% до Фонду соц. страхування безробіття, 1% - страхування при втраті працездатності

double taxIncome = grossSalary \* 0.15;

double taxPension = grossSalary \* 0.02;

double taxUnemployment = grossSalary \* 0.006;

double taxTemporary = grossSalary \* 0.01;

double totalTaxes = taxIncome + taxPension + taxUnemployment + taxTemporary;

double netSalary = grossSalary - totalTaxes;

SalaryInfo info = { grossSalary, totalTaxes, netSalary };

return info;

}

// Функція convertSockSize приймає розмір в українській системі і повертає відповідники для США та ЄС

SockSize convertSockSize(int size\_UA) {

SockSize result;

result.valid = true;

if (size\_UA == 23) {

result.size\_US = 8;

result.size\_EU = "37/38";

}

else if (size\_UA == 25) {

result.size\_US = 9;

result.size\_EU = "39/40";

}

else if (size\_UA == 27) {

result.size\_US = 10;

result.size\_EU = "41/42";

}

else if (size\_UA == 29) {

result.size\_US = 11;

result.size\_EU = "43/44";

}

else if (size\_UA == 31) {

result.size\_US = 12;

result.size\_EU = "45/46";

}

else {

// Якщо розмір не належить до визначеної таблиці - прапорець valid встановлюємо в false

result.valid = false;

result.size\_US = -1;

result.size\_EU = "";

}

return result;

}

int countBinarySymbols(unsigned int N) {

// Отримання 32-бітного рядка з двійковим представленням числа N

std::string binaryStr = std::bitset<32>(N).to\_string();

// Обрізаємо ведучі нулі, щоб аналізувати лише значущу частину

size\_t pos = binaryStr.find('1');

if (pos != std::string::npos) {

binaryStr = binaryStr.substr(pos);

} else {

// Якщо число 0, то встановлюємо рядок "0"

binaryStr = "0";

}

// Перевірка молодшого біту (LSB)

bool leastSignificantBit = (N & 1);

// Використовуємо тернарний оператор для визначення, який символ рахувати:

// якщо молодший біт = 0 – рахуємо '0', інакше – '1'.

char targetChar = (leastSignificantBit == 0) ? '0' : '1';

// Підрахунок символів targetChar у обрізаному рядку

int count = std::count(binaryStr.begin(), binaryStr.end(), targetChar);

return count;

}

# Вихідний код TestDriver:

#include <iostream>

#include <iomanip>

#include <cmath>

#include "ModulesMarchenko.h"

// Функція для порівняння чисел з плаваючою комою з точністю ε

bool areEqual(double a, double b, double epsilon = 0.001) {

return std::fabs(a - b) < epsilon;

}

// Функція тестування задачі 9.1

void testTask9\_1() {

std::cout << "=== Тестування задачі 9.1: Зарплата та податки ===" << std::endl;

bool allPassed = true;

// Тест-кейс 1: Кількість днів = 0

{

int daysWorked = 0;

SalaryInfo result = calculateSalary(daysWorked);

double expGross = 0;

double expTaxes = 0;

double expNet = 0;

bool passed = areEqual(result.grossSalary, expGross) &&

areEqual(result.totalTaxes, expTaxes) &&

areEqual(result.netSalary, expNet);

std::cout << "[TC1] Вхід: daysWorked = " << daysWorked << std::endl;

std::cout << " Отримано: grossSalary = " << result.grossSalary

<< ", totalTaxes = " << result.totalTaxes

<< ", netSalary = " << result.netSalary << std::endl;

std::cout << " Очікувано: " << expGross << ", " << expTaxes << ", " << expNet

<< " -> " << (passed ? "PASSED" : "FAILED") << std::endl << std::endl;

allPassed = allPassed && passed;

}

// Тест-кейс 2: Кількість днів = 5

{

int daysWorked = 5;

SalaryInfo result = calculateSalary(daysWorked);

double expGross = 5 \* 8 \* 450; // 18000 грн

double expTaxes = expGross \* 0.15 + // 15%

expGross \* 0.02 + // 2%

expGross \* 0.006 + // 0.6%

expGross \* 0.01; // 1%

double expNet = expGross - expTaxes; // 18000 - 3348 = 14652 грн (заокруглено)

bool passed = areEqual(result.grossSalary, expGross) &&

areEqual(result.totalTaxes, expTaxes) &&

areEqual(result.netSalary, expNet);

std::cout << "[TC2] Вхід: daysWorked = " << daysWorked << std::endl;

std::cout << " Отримано: grossSalary = " << result.grossSalary

<< ", totalTaxes = " << result.totalTaxes

<< ", netSalary = " << result.netSalary << std::endl;

std::cout << " Очікувано: " << expGross << ", " << expTaxes << ", " << expNet

<< " -> " << (passed ? "PASSED" : "FAILED") << std::endl << std::endl;

allPassed = allPassed && passed;

}

// Тест-кейс 3: Кількість днів = 10

{

int daysWorked = 10;

SalaryInfo result = calculateSalary(daysWorked);

double expGross = 10 \* 8 \* 450; // 36000 грн

double expTaxes = expGross \* 0.15 + // 15%

expGross \* 0.02 + // 2%

expGross \* 0.006 + // 0.6%

expGross \* 0.01; // 1%

double expNet = expGross - expTaxes; // 36000 - 6696 = 29304 грн (заокруглено)

bool passed = areEqual(result.grossSalary, expGross) &&

areEqual(result.totalTaxes, expTaxes) &&

areEqual(result.netSalary, expNet);

std::cout << "[TC3] Вхід: daysWorked = " << daysWorked << std::endl;

std::cout << " Отримано: grossSalary = " << result.grossSalary

<< ", totalTaxes = " << result.totalTaxes

<< ", netSalary = " << result.netSalary << std::endl;

std::cout << " Очікувано: " << expGross << ", " << expTaxes << ", " << expNet

<< " -> " << (passed ? "PASSED" : "FAILED") << std::endl << std::endl;

allPassed = allPassed && passed;

}

std::cout << "Результат тестування задачі 9.1: " << (allPassed ? "ВСІ ТЕСТИ PASSED" : "Є FAILED тести") << std::endl << std::endl;

}

// Функція тестування задачі 9.2

void testTask9\_2() {

std::cout << "=== Тестування задачі 9.2: Конвертація розміру шкарпеток ===" << std::endl;

bool allPassed = true;

// Тест-кейс 1: UA розмір = 23, очікувано: США 8, ЄС "37/38"

{

int uaSize = 23;

SockSize result = convertSockSize(uaSize);

bool passed = result.valid && (result.size\_US == 8) && (result.size\_EU == "37/38");

std::cout << "[TC1] Вхід: uaSize = " << uaSize << std::endl;

std::cout << " Отримано: US = " << result.size\_US

<< ", EU = " << result.size\_EU << std::endl;

std::cout << " Очікувано: US = 8, EU = 37/38 -> " << (passed ? "PASSED" : "FAILED") << std::endl << std::endl;

allPassed = allPassed && passed;

}

// Тест-кейс 2: UA розмір = 27, очікувано: США 10, ЄС "41/42"

{

int uaSize = 27;

SockSize result = convertSockSize(uaSize);

bool passed = result.valid && (result.size\_US == 10) && (result.size\_EU == "41/42");

std::cout << "[TC2] Вхід: uaSize = " << uaSize << std::endl;

std::cout << " Отримано: US = " << result.size\_US

<< ", EU = " << result.size\_EU << std::endl;

std::cout << " Очікувано: US = 10, EU = 41/42 -> " << (passed ? "PASSED" : "FAILED") << std::endl << std::endl;

allPassed = allPassed && passed;

}

// Тест-кейс 3: UA розмір = 31, очікувано: США 12, ЄС "45/46"

{

int uaSize = 31;

SockSize result = convertSockSize(uaSize);

bool passed = result.valid && (result.size\_US == 12) && (result.size\_EU == "45/46");

std::cout << "[TC3] Вхід: uaSize = " << uaSize << std::endl;

std::cout << " Отримано: US = " << result.size\_US

<< ", EU = " << result.size\_EU << std::endl;

std::cout << " Очікувано: US = 12, EU = 45/46 -> " << (passed ? "PASSED" : "FAILED") << std::endl << std::endl;

allPassed = allPassed && passed;

}

std::cout << "Результат тестування задачі 9.2: " << (allPassed ? "ВСІ ТЕСТИ PASSED" : "Є FAILED тести") << std::endl << std::endl;

}

// Функція тестування задачі 9.3

void testTask9\_3() {

std::cout << "=== Тестування задачі 9.3: Підрахунок бінарних символів ===" << std::endl;

bool allPassed = true;

// Тест-кейс 1: N = 0, двійкове представлення "0" → очікується 1 нуль

{

unsigned int N = 0;

int result = countBinarySymbols(N);

int exp = 1; // Для "0" маємо один символ '0'

bool passed = (result == exp);

std::cout << "[TC1] Вхід: N = " << N << std::endl;

std::cout << " Отримано: " << result << std::endl;

std::cout << " Очікувано: " << exp << " -> " << (passed ? "PASSED" : "FAILED") << std::endl << std::endl;

allPassed = allPassed && passed;

}

// Тест-кейс 2: N = 6, двійкове представлення "110"; молодший біт = 0, підрахунок нулів має дати 1

{

unsigned int N = 6; // binary "110"

int result = countBinarySymbols(N);

int exp = 1; // лише одна '0'

bool passed = (result == exp);

std::cout << "[TC2] Вхід: N = " << N << std::endl;

std::cout << " Отримано: " << result << std::endl;

std::cout << " Очікувано: " << exp << " -> " << (passed ? "PASSED" : "FAILED") << std::endl << std::endl;

allPassed = allPassed && passed;

}

// Тест-кейс 3: N = 7, двійкове представлення "111"; молодший біт = 1, підрахунок одиниць має дати 3

{

unsigned int N = 7; // binary "111"

int result = countBinarySymbols(N);

int exp = 3; // усі 3 біти - '1'

bool passed = (result == exp);

std::cout << "[TC3] Вхід: N = " << N << std::endl;

std::cout << " Отримано: " << result << std::endl;

std::cout << " Очікувано: " << exp << " -> " << (passed ? "PASSED" : "FAILED") << std::endl << std::endl;

allPassed = allPassed && passed;

}

std::cout << "Результат тестування задачі 9.3: " << (allPassed ? "ВСІ ТЕСТИ PASSED" : "Є FAILED тести") << std::endl << std::endl;

}

int main() {

std::cout << "=== Тестовий драйвер для модульного тестування задач 9.1, 9.2 та 9.3 ===" << std::endl << std::endl;

// Тестування кожного модуля з використанням розроблених наборів контрольних прикладів

testTask9\_1();

testTask9\_2();

testTask9\_3();

std::cout << "=== Завершення тестування ===" << std::endl;

return 0;

}

# ЗАДАЧА 9.4

**Алгоритм:**

1. **Ініціалізація та організація циклу:**

* Запустити нескінченний цикл, поки не буде введено символ для виходу.

1. **Основний блок обробки введення:**

* Запитати користувача про введення символу.
* Прочитати символ.
* Обробити введений символ за допомогою конструкції switch чи if-else:
* Якщо символ дорівнює **'v'**:
  + Викликати функцію **s\_calculation()**.
* Якщо символ дорівнює **'n'**:
  + Викликати функцію задачі 9.1.
* Якщо символ дорівнює **'m'**:
  + Викликати функцію задачі 9.2.
* Якщо символ дорівнює **'q'**:
  + Викликати функцію задачі 9.3.
* Для інших символів:
  + Вивести повідомлення про помилкове введення і здійснити звуковий сигнал (вивести символ «\a»).

1. **Запит про завершення роботи:**

* Після виконання функції виведіть повідомлення з запитом “Для виходу введіть 'w' або 'W', або будь-який інший символ для продовження”.
* Прочитати введення користувача.
* Якщо введено символ **'w'** або **'W'**, вийти з циклу (завершити програму).
* Інакше – повернутися до початку циклу.

1. **Протоколювання:**

* Під час кожного кроку виводити на екран:
* Вхідний символ з описом (наприклад, "Введено: 'm' – виклик функції задачі 9.2").
* Результат виконання викликаної функції.
* Повідомлення про помилкове введення (якщо застосовано).
* Повідомлення про стан завершення циклу (чи продовжується робота, чи завершується програма).

# Вихідний код Marchenko\_task:

#include <iostream>

#include "ModulesMarchenko.h"

using namespace std;

int main() {

char command; // Символ, що задає вибір функції

char exitCommand; // Символ для виходу з програми

do {

// Виведення меню вибору

cout << "========================================" << endl;

cout << "Введіть команду для виклику функції:" << endl;

cout << " v - s\_calculation()" << endl;

cout << " n - задачa 9.1 (calculateSalary)" << endl;

cout << " m - задачa 9.2 (convertSockSize)" << endl;

cout << " q - задачa 9.3 (countBinarySymbols)" << endl;

cout << "========================================" << endl;

cout << "Ваш вибір: ";

cin >> command;

cout << endl;

switch (command) {

case 'v': {

double x, y, z;

cout << "Введіть x, y, z: ";

cin >> x >> y >> z;

cout << "Виклик функції s\_calculation():" << endl;

cout << s\_calculation(x, y, z) << endl;

break;

}

case 'n': {

// Виклик задачі 9.1: розрахунок зарплати, податків та чистої виплати

cout << "Виклик функції calculateSalary()." << endl;

cout << "Введіть кількість відпрацьованих днів: ";

int daysWorked;

cin >> daysWorked;

SalaryInfo salary = calculateSalary(daysWorked);

cout << "Нарахована зарплата: " << salary.grossSalary << " грн" << endl;

cout << "Податки: " << salary.totalTaxes << " грн" << endl;

cout << "Чиста зарплата: " << salary.netSalary << " грн" << endl;

break;

}

case 'm': {

// Виклик задачі 9.2: конвертація розмірів шкарпеток

cout << "Виклик функції convertSockSize()." << endl;

cout << "Введіть розмір шкарпеток (український розмір): ";

int sizeUA;

cin >> sizeUA;

SockSize sock = convertSockSize(sizeUA);

if (!sock.valid) {

cout << "Некоректний розмір шкарпеток!" << endl;

} else {

cout << "Розмір для США: " << sock.size\_US << endl;

cout << "Розмір для ЄС: " << sock.size\_EU << endl;

}

break;

}

case 'q': {

// Виклик задачі 9.3: підрахунок кількості бінарних символів

cout << "Виклик функції countBinarySymbols()." << endl;

cout << "Введіть натуральне число: ";

unsigned int N;

cin >> N;

int count = countBinarySymbols(N);

// Виводимо інформацію залежно від молодшого біта

if ((N & 1) == 0)

cout << "Молодший біт = 0, кількість двійкових нулів: " << count << endl;

else

cout << "Молодший біт = 1, кількість двійкових одиниць: " << count << endl;

break;

}

default:

// Якщо введено некоректний символ – повідомлення про помилку з звуковим сигналом

cout << "Некоректне введення! " << "\a" << endl;

break;

}

// Запит чи бажає користувач вийти

cout << endl << "Для виходу введіть 'w' або 'W', або будь-який інший символ для продовження: ";

cin >> exitCommand;

cout << endl;

} while (exitCommand != 'w' && exitCommand != 'W');

cout << "Програма завершена." << endl;

return 0;

}

# Аргументи досягнення мети:

* + - 1. **Реалізація розгалуженої логіки:** Використання умовних операторів (if-else, switch) у рішеннях задач 9.1–9.3 дозволило правильно розподілити логіку виконання в залежності від вхідних параметрів.
      2. **Коректне обчислення зарплати:** Функція задачі 9.1 точно розраховує нараховану зарплату на основі кількості відпрацьованих днів, що є ключовим для валідації алгоритму.
      3. **Підрахунок податків:** Правильний розподіл відрахувань (15%, 2%, 0.6%, 1%) свідчить про детальне врахування вимог до обчислення.
      4. **Використання структур:** Реалізація результатів у вигляді структур (SalaryInfo, SockSize) покращує читаність та модульність коду.
      5. **Обґрунтований розподіл задач:** Кожна задача (9.1, 9.2, 9.3) реалізована у окремій функції, що відповідає принципу модульності та спрощує тестування.
      6. **Використання статичної бібліотеки:** Інтеграція функцій у статичну бібліотеку libModulesПрізвище.a дозволяє повторно використовувати код у різних проектах.
      7. **Чіткий інтерфейс:** Заголовковий файл ModulesMarchenko.h забезпечує однозначність виклику функцій, що сприяє легшій інтеграції та розширенню ПЗ.
      8. **Застосування тернарного оператора:** При реалізації задачі 9.3 використання тернарного оператора дозволило лаконічно обрати спосіб підрахунку символів (нуля чи одиниці).
      9. **Обрізання ведучих нулів:** Модифікація функції countBinarySymbols із обрізанням ведучих нулів покращує точність обчислень для числа 0 та інших випадків.
      10. **Адаптивність до різних платформ:** Використання кросплатформового середовища (Code::Blocks) дозволяє запускати проекти на Windows, Linux та Mac OS X.
      11. **Повторне використання коду:** Реалізація окремих функцій для кожної задачі дозволяє повторно використовувати їх у різних програмних модулях.
      12. **Підтримка модульного тестування:** Розроблений тестовий драйвер та тест-сьют забезпечують можливість автоматичного тестування кожної функції окремо.
      13. **Документоване протоколювання:** Протоколювання виведення вхідних даних і результатів дозволяє аналізувати хід виконання тестів і виявляти можливі помилки.
      14. **Валідація вхідних даних:** Реалізація перевірки коректності введених розмірів шкарпеток (за допомогою прапорця valid) свідчить про надійність рішення.
      15. **Інтерактивне меню:** Завдання 9.4 реалізовано як інтерактивне меню, що забезпечує зручний вибір користувача для виклику відповідних функцій.
      16. **Обробка некоректного введення:** Наявність обробки некоректних команд (через default у switch) із звуковим сигналом покращує зручність та надійність інтерфейсу.
      17. **Функціональна декомпозиція:** Розподіл завдань за окремими функціями полегшує налагодження та подальше розширення функціоналу.
      18. **Чітка структура коду:** Розділення коду на логічні блоки (відповідно до задач 9.1–9.4) спрощує читання й розуміння реалізації.
      19. **Модульність тестування:** Окремий проект TestDriver дозволяє ізолювати та перевіряти окремі модулі, що гарантує правильність роботи ПЗ.
      20. **Використання бібліотек C++ STL:** Застосування стандартних бібліотечних алгоритмів, таких як std::count, забезпечує ефективність і надійність обчислень.
      21. **Легкість масштабування:** Модульна структура дозволяє додавати нові задачі або функції, не змінюючи основної архітектури.
      22. **Зрозумілі коментарі:** Наявність коментарів у коді допомагає швидко орієнтуватися в алгоритмах і полегшує подальшу підтримку.
      23. **Покриття вимог технічного завдання:** Реалізація відповідає вимогам методички, включно з інтерактивним вибором та можливістю тестування.
      24. **Підтримка відлагодження:** Логування результатів роботи функцій сприяє швидкому виявленню і виправленню помилок.
      25. **Коректна реалізація розгалужень:** Перевірка введення символів забезпечує коректне розгалуження логіки, що відповідає методичним рекомендаціям.
      26. **Валідація даних при виконанні арифметичних операцій:** Розрахунок зарплати і податків показує правильне використання арифметичних операторів з плаваючою комою.
      27. **Ефективність алгоритмів:** Використання оптимізованих алгоритмів (наприклад, std::bitset для двійкового представлення) забезпечує швидке виконання.
      28. **Відповідність принципам ООП:** Використання структур для зберігання даних відповідає принципам організації програмних модулів.
      29. **Інкапсуляція логіки:** Логіка кожного з завдань закладена у власну функцію, що гарантує інкапсуляцію і зменшує взаємозалежність між модулями.
      30. **Підвищення надійності ПЗ:** Використання обробки помилок і перевірок вводу мінімізує ризик непередбачуваної поведінки при некоректних даних.
      31. **Зручність для користувача:** Інтерактивне меню із зрозумілими підказками забезпечує позитивний досвід користування програмою.
      32. **Гнучкість реалізації:** Функції можуть бути викликані як поодинці, так і у складі інтегрованого застосунку, що підвищує їх універсальність.
      33. **Можливість інтеграції з іншими проектами:** Статична бібліотека легко інтегрується в інші проекти, що підвищує її практичну цінність.
      34. **Відповідність стандартам кодування:** Використання заголовочних файлів, структурування коду та коментарі відповідають вимогам сучасного програмування.
      35. **Підтримка модульної розробки:** Чітке розділення функцій дозволяє відокремити логіку обчислень від інтерфейсу користувача.
      36. **Покращення навичок програмування:** Виконання завдань сприяє розширенню практичних навичок у використанні розгалужених та ітераційних конструкцій.
      37. **Практична орієнтація:** Лабораторна робота допомагає засвоїти методи розробки реальних програмних модулів для обчислювальних процесів.
      38. **Логічна побудова алгоритмів:** Прецизійно сформульовані алгоритми для кожної задачі допомагають забезпечити правильність та передбачуваність результатів.
      39. **Точність обчислень:** Аритметичні операції виконуються з високою точністю, що особливо важливо для фінансових розрахунків.
      40. **Управління потоками виконання:** Використання циклів для повторення запитів та вибору дій забезпечує контроль над виконанням програми.
      41. **Чіткість вибору дій:** Меню дозволяє користувачу легко орієнтуватися між доступними варіантами, що підвищує інтуїтивність застосунку.
      42. **Динамічність роботи:** Можливість повторного виклику функцій без перезапуску програми забезпечує гнучкість застосунку.
      43. **Підвищення ефективності налагодження:** Зрозумілі повідомлення про помилки та логування вхідних параметрів дозволяють швидко знаходити недоліки.
      44. **Компактність реалізації:** Код написано лаконічно, без надлишкових конструкцій, що сприяє його легкому аналізу.
      45. **Відповідність методичним рекомендаціям:** Виконання завдань строго відповідає вказівкам методички, що засвідчує повноту розуміння поставлених завдань.
      46. **Досвід роботи з Git-репозиторієм:** Використання Git-репозиторію для збереження проектів показує уміння організувати робочий процес і контролювати версії.
      47. **Орієнтація на кросплатформність:** Проект побудовано із врахуванням роботи на різних ОС, що підвищує його універсальність.
      48. **Інформативність звіту:** Докладне описання всіх кроків розробки і тестування забезпечує повний звіт про виконану роботу.
      49. **Практична цінність отриманих знань:** Завдання дозволяють застосувати теоретичні знання на практиці, що є ключовим для професійного зростання.
      50. **Сумісність з сучасними стандартами:** Реалізація за останніми стандартами С++ забезпечує тривалий термін використання та легке впровадження удосконалень.