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.

ВИКОНАВ:
студент академічної групи КН-23
Клюй А.Я.

ПЕРЕВІРИВ:
викладач кафедри кібербезпеки
та програмного забезпечення
Дрєєва Г.М.

Кропивницький - 2024

МЕТА: Набути ґрунтовних вмінь і практичних навичок реалізації технології модульного програмування, застосування операторів C/C++ 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.a (проект ModulesПрізвище, створений під час виконання лабораторної роботи No 8).

2. Реалізувати програмне забезпечення розв'язування задачі 9.4 на основі функцій статичної бібліотеки libModulesПрізвище.a.

ВАРІАНТ № 5

ЗАДАЧА 9.1

У супермаркеті за кожну покупку на суму від 100 до 200 грн. на накопичувальну картку нараховується 1 бонус, від 200 до 500 грн. - 5 бонусів, від 200 до 500 грн. — 7 бонусів, від 500 до 1000 грн. - 10 бонусів, від 1000 до 2500 грн. - 50 бонусів, від 2500 до 5000 грн. - 150 бонусів, понад 5000 грн. - 300 бонусів.

Вхід: сума покупки, грн.

Вихід: кількість нарахованих бонусів за покупку та сума до сплати з врахуванням знижки у грн. (1 бонус - 25 коп. знижки).

ЗАДАЧА 9.2

Вхід: кількість градусів за шкалою Фаренгейта.

Вихід: конвертовані градуси у шкалу Цельсія.

Знаючи температуру за шкалою Фаренгейта, температура за шкалою Цельсія розраховується наступним чином:

$$t_C = \frac{5}{9}(t_F - 32)$$

де $t_c$ - температура за шкалою Цельсія, $t_F$ - температура за шкалою Фаренгейта.

ЗАДАЧА 9.3

Вхід: натуральне число N від 0 до 51950.

Вихід: якщо біт D числа N рівний 0, кількість двійкових нулів у ньому, інакше — кількість двійкових одиниць*

*під час підрахунку кількості бінарних 0 або 1 рекомендовано використати тернарний оператор « ? : ».

ЗАДАЧА 9.4

За введеним користувачем символом "j" викликається `s_calculation ()`, "z" - функція задачі 9.1, "x" - функція задачі 9.2, "c" - функція задачі 9.3; якщо користувач вводить інші символи, вони ігноруються, при чому видається звуковий сигнал про помилкове введення. Після цього, якщокористувач за запитом додатка вводить символ "v", "V" або "A", відбувається вихід з програми, інакше — виконання програми повторюється.

## ХІД РОБОТИ

ЗАДАЧА 9.1

**Аналіз:**

1. Умови нарахування бонусів
2. Розрахунок знижки за бонуси:
3. Вихідні дані.

**Постановка задачі**

Розробити програму, яка на вхід отримує суму покупки в гривнях і вираховує:

- Кількість нарахованих бонусів за дану покупку.
- Суму до сплати з врахуванням знижки

**Аналіз вимог:**

1. Програма має приймати вхідні дані у вигляді суми покупки в гривнях.

2. В залежності від суми покупки, програма повинна нараховувати бонусні бали.

3. На основі нарахованих бонусів, програма повинна вирахувати суму знижки в гривнях (де 1 бонус = 25 копійок знижки).

4. Вихідними даними програми є кількість бонусів та сума до сплати з врахуванням знижки.

**Проектування архітектури**

Програму можна розробити як консольний застосунок. Вона має наступні складові:

1. Модуль введення даних: забезпечує отримання вхідної суми покупки.

2. Модуль обчислень: виконує логіку нарахування бонусів та розрахунок знижки.

3. Модуль виведення даних: показує результат користувачу.

**Детальне проектування програмних модулів:**

1. Модуль введення даних:

- Вхід: користувач вводить суму покупки.
- Перевірка: програма перевіряє, чи сума є дійсним числом у діапазоні допустимих значень.
- Вихід: сума покупки передається до модуля обчислень.

2. Модуль обчислень:

- Вхід: сума покупки від модуля введення даних.
- Визначити діапазон, до якого належить сума покупки.
- Нарахувати відповідну кількість бонусів згідно з правилами.
- Перевести бонуси у суму знижки (1 бонус = 25 копійок).
- Вихід: кількість бонусів та сума до сплати з урахуванням знижки.

3. Модуль виведення даних:

- Вхід: кількість бонусів і сума до сплати від модуля обчислень.
- Виконання: відображення результату користувачу у зрозумілому форматі.
- Вихід: немає.

Програма має включати обробку помилок для випадків введення недійсних даних і забезпечувати зручний інтерфейс для користувача. Для консольної програми можуть бути задіяні такі мови програмування як C/C++

ЗАДАЧА 9.2

**Аналіз:**

- Виконати конвертацію температури з градусів за шкалою Фаренгейта в градуси за шкалою Цельсія.
- Створити функцію для перетворення температури з Фаренгейта в Цельсія.

**Постановка задачі:**

Створити програму або алгоритм, який приймає на вхід значення температури в градусах за шкалою Фаренгейта, перетворює його в градуси за шкалою Цельсія за допомогою вищенаведеної формули та виводить результат.

**Розробити програму, яка:**

- Отримує на вхід значення температури в градусах Фаренгейта.
- Перетворює введене значення в градуси Цельсія за допомогою наведеної формули.
- Виводить результат перетворення.

**Аналіз вимог:**

Ця задача вимагає конвертації температури з Фаренгейтів в Цельсії за заданою формулою. Програма повинна приймати вхідними даними температуру в градусах за шкалою Фаренгейта і виводити результат в градусах за шкалою Цельсія.

**Вимоги до програми:**

1. Вхідні дані: температура в градусах Фаренгейта (можливо дробове число).
2. Обчислення: використання формули перетворення температури.
3. Вихідні дані: температура в градусах Цельсія (можливо дробове число).

**Проектування архітектури**

Програму можна розробити як консольний застосунок. Вона має наступні складові:

1. Модуль введення даних: інтерфейс для вводу температури в градусах Фаренгейта.
2. Модуль обчислень: функція для перетворення температури.
3. Модуль виведення даних: інтерфейс для відображення результату конвертації.

**Детальне проектування програмних модулів:**

1. Модуль введення даних:
   - Функція для введення температури користувачем.
   - Перевірка на коректність введених даних (число, можливо в діапазоні реальних значень температур).
   - Передача даних до модуля обчислень.
2. Модуль обчислень:
   - Функція конвертації, яка використовує формулу $t_C = \frac{5}{9}(t_F - 32)$.
- Передача результату до модуля виведення даних.
3. Модуль виведення даних:
   - Відображення сконвертованої температури в Цельсіях.
   - Забезпечення можливості повторного вводу або закінчення роботи програми.

ЗАДАЧА 9.3

**Аналіз:**

- Задача стосується обробки бітів натурального числа N.
- Перевірити значення четвертого біта від кінця (D4) двійкового представлення числа N.
- В залежності від того, чи дорівнює цей біт нулю, потрібно підрахувати кількість двійкових нулів або кількість двійкових одиниць в усьому числі N.

**Постановка задачі**

Розробити алгоритм або програму, яка:

- Отримує на вхід натуральне число N ($0 \leq N \leq 51950$).
- Визначає значення четвертого біта з кінця (D4) у двійковому представленні числа N.
- В залежності від значення D4, підраховує кількість двійкових нулів або одиниць в числі N.
- Виводить результат підрахунку.

**Аналіз вимог:**

Задача вимагає обробки натурального числа (N), заданого в діапазоні від 0 до 51950. Необхідно визначити значення четвертого біта з кінця (D4) у двійковому представленні числа (N) і, залежно від його значення, підрахувати кількість двійкових нулів або одиниць у всьому числі.

**Вимоги до програми:**

1. Вхідні дані: натуральне число (N) в діапазоні від 0 до 51950.
2. Обчислення: перевірка четвертого біта з кінця і підрахунок двійкових нулів або одиниць.
3. Вихідні дані: кількість двійкових нулів або одиниць, залежно від значення біта D4.

**Проектування архітектури**

Програму можна розробити як консольний застосунок. Вона має наступні складові:

1. Модуль введення даних: отримання та валідація вхідного числа.
2. Модуль логіки обчислень: визначення біта D4 і підрахунок необхідних бітів.
3. Модуль виведення даних: відображення результату користувачу.

**Детальне проектування програмних модулів:**

1. Модуль введення даних:

• Функція `natural_number` для зчитування вхідного числа від користувача.

• Валідація введеного числа: перевірка на те, що є натуральним числом і належить діапазону від 0 до 51950.

• Передача числа модулю обчислень.

2. Модуль обчислень:

• Функція `calculate_number` приймає число як параметр.

• Використання бітових операцій для визначення стану четвертого біта з кінця.

• Переведення числа у двійкову форму і підрахунок нулів або одиниць залежно від стану біта D4.

• Тернарний оператор «?»: для вибору між підрахунком нулів або одиниць.

• Повернення результату підрахунку.

3. Модуль виведення даних:

• Функція `output_result` приймає кількість нулів або одиниць і стан біта D4.

• Виведення повідомлення з кількістю підрахованих нулів або одиниць, згідно зі станом біта D4.

ЗАДАЧА 9.4

**Аналіз задачі**

Вхідні дані:

- Символ, введений користувачем

Вихідні дані:

- Виконання певної функції відповідно до введеного символу
- Звуковий сигнал у випадку неправильного введення

**Основні кроки програми:**

- Введення символу користувачем
- Виклик відповідної функції в залежності від введеного символу
- У випадку неправильного введення - відтворення звукового сигналу

- Можливість повторення програми або виходу з неї в разі введення символів "v", "V" або "A"

**Постановка задачі:**

- Написати програму, яка надає користувачу можливість викликати різні функції згідно з введеним символом
- При введенні символу "j" викликати функцію задачі 8.1, `s_calculation()`.
- При введенні символу "z" викликати функцію задачі 9.1.
- При введенні символу "x" викликати функцію задачі 9.2.
- При введенні символу "c" викликати функцію задачі 9.3
- У випадку введення інших символів, відтворити звуковий сигнал про помилкове введення
- Після кожного виконання функції дати користувачеві можливість вибору: повторити програму або вийти з неї
- У випадку введення символів "v", "V" або "A" завершити програму.

**ModulesKlui:**

```
#include <math.h>
#include "ModulesKlui.h"
#include <iostream>
#ifdef __APPLE__
#   include <AppKit/AppKit.h>
#endif

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


// Ввести сумму покупки в гривнях
int read_data (){
    int x;
    std::cout << "Введіть суму покупки в гривнях: ";
    std::cin >> x;
    return x;
}
```

```
// Вивести результат
void output_result (int x, double y){
    std::cout << "Кількість нарахованих бонусів - " <<
    x << ", за покупку та сума до сплати з врахуванням знижки у гривнях - " << y
<< std::endl;
}

// Вираховування кількості бонусів
int number_bonus (int x){
    int bonus = 0;
    if (x >= 100 && x < 200){
        bonus = 1;
    } else if (x >= 200 && x < 500){
        bonus = 5;
    } else if (x >= 500 && x < 1000){
        bonus = 10;
    } else if (x >= 1000 && x < 2500){
        bonus = 50;
    } else if (x >= 2500 && x < 5000){
        bonus = 150;
    } else if (x >= 5000){
        bonus = 300;
    }
    return bonus;
}

// Конвертування бонусів в знижку
double discount_bonuses (int bonus){
    return bonus * 0.25;
}



// Функція конвертування t градусів з Фіренгейта у Цельсія
double degree_conversion(double f) {
    return (5.0 / 9.0) * (f - 32.0);
}

double degree_fahrenheit (){
    double f;
    std::cout << "Введіть кількість градусів Фіренгейта: ";
    std:: cin >> f;
    return f;
}
```

```
void degrees_celsius (double f){
    std::cout << "Градуси у шкалі цельсія: " << f << std::endl;
}


// Функція підрахунку кількості бінарних 0 або 1 у числі N
unsigned short calculate_number(unsigned short n){
    auto counter = 0;
    bool k = (n & 0b10000) == 0b10000;
    for(auto i = 0; i < 16 ; i++) {
      if ((n & 1) == 1) {
          counter++;
      }
    n = n >> 1;
    }
    if (k) {
       return counter;
    }
    return 16 - counter;
}

unsigned short natural_number(){
    unsigned short N;
    std::cout << "Введіть число від 0 до 51950: ";
    std:: cin >> N;
    return N;
}

void output_result(unsigned short n) {
    std::cout << "Результат: " << n << std::endl;
}


int last() {
    char choice;

    do {
        std::cout << "Введіть символ j, z, x, c або v, V, A для виходу: ";
        std::cin >> choice;

        // Перевірка введеного символу
        switch (choice) {
            case 'j':
```

```
                task_8_1();
                break;
            case 'z':
                task_9_1();
                break;
            case 'x':
                task_9_2();
                break;
            case 'c':
                task_9_3();
                break;
            case 'v':
                // Вихід з програми
                std::cout << "Дякую за використання програми. До побачення!\n";
                exit(0);
            default:
                // Відтворення звукового сигналу про помилку
#ifdef __APPLE__
                NSBeep(); // звуковий сигнал про помилку на macOS
#else
                std::cout << '\a'; // звуковий сигнал про помилку
#endif
                std::cout << "Неправильне введення. Спробуйте ще раз.\n";
        }
    } while (true);

    return 0;
}


void task_9_1() {

    int total_sum = read_data();
    int total_bonus = number_bonus(total_sum);
    double total_discount = discount_bonuses(total_bonus);
    output_result(total_bonus, total_sum - total_discount);
}


void task_9_2() {
    double f = degree_fahrenheit();
    double c = degree_conversion(f);
    degrees_celsius(c);
}
```

```
void task_9_3() {
    last();
}


void  task_8_1() {
    // 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;
}
```

## TestDriver:

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

int main() {

    int total_sum = read_data();
    int total_bonus = number_bonus(total_sum);
    double total_discount = discount_bonuses(total_bonus);
    output_result(total_bonus, total_sum - total_discount);

    return 0;
}
```

## TestDriver2:

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

```
int main() {
    double f = degree_fahrenheit();
    double c = degree_conversion(f);
    degrees_celsius(c);
}
```

**TestDriver3:**

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

int main(){
    unsigned short N = natural_number();
    unsigned short n = calculate_number(N);
    output_result(n);
}
```

**TestDriver4:**

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

int main() {
    last();
    return 0;
}
```

**TestCase_task9.4:**

TC-01: Введіть символ j, z, x, c або v, V, A для виходу: j

Enter x: 2

Enter y: 3

Enter z: 5

S = 222.631

TC-02: Введіть символ j, z, x, c або v, V, A для виходу: z

Введіть суму покупки в гривнях: 350

Кількість нарахованих бонусів - 5, за покупку та сума до сплати з врахуванням знижки у гривнях - 348.75

TC-03: Введіть символ j, z, x, c або v, V, A для виходу: x

Введіть кількість градусів Фіренгейта: 41

Градуси у шкалі цельсія: 5

ТС-04: Введіть символ j, z, x, c або v, V, A для виходу: s

Неправильне введення. Спробуйте ще раз.

ТС-05: Введіть символ j, z, x, c або v, V, A для виходу: a

Неправильне введення. Спробуйте ще раз.

ТС-06: Введіть символ j, z, x, c або v, V, A для виходу: v

Дякую за використання програми. До побачення!

ТС-07: Введіть символ j, z, x, c або v, V, A для виходу: V

Неправильне введення. Спробуйте ще раз.

ТС-08: Введіть символ j, z, x, c або v, V, A для виходу: A

Неправильне введення. Спробуйте ще раз.

ТС-09: Введіть символ j, z, x, c або v, V, A для виходу: m

Неправильне введення. Спробуйте ще раз.

ТС-10: Введіть символ j, z, x, c або v, V, A для виходу: k

Неправильне введення. Спробуйте ще раз

## ВИСНОВКИ

Хід роботи був спрямований на вирішення задачі 9.1, яка полягає у розрахунку кількості нарахованих бонусів та знижки для покупок у супермаркеті в залежності від їхньої суми. Виконавши аналіз вимог, була розроблена структура програми, в якій відведено окремі функції для введення даних, розрахунку кількості бонусів та знижки, а також виведення результату. Для зручності програма була розділена на модулі, що дозволило забезпечити її легку розширюваність та обслуговуваність. Були реалізовані функції для введення суми покупки, розрахунку кількості нарахованих бонусів та знижки, відповідно до вимог задачі 9.1. Реалізація функцій була здійснена мовою програмування C++, забезпечуючи ефективну та швидку роботу програми. Було реалізовано тестовий драйвер для модульного тестування функцій розв'язування цієї задачі. Проведено модульне тестування, що дозволило перевірити правильність роботи функцій та впевнитися у їхній надійності та ефективності.

Отриманні такі артефакти:

1) модуль введення даних: `read_data()`- функція введення суми покупки.

2) модуль розрахунку кількості бонусів: `number_bonus(int x)`- функція розрахунку кількості нарахованих бонусів.

3) модуль конвертації бонусів у знижку: `discount_bonuses(int bonus)`-функція конвертації бонусів у знижку.

4) модуль виведення результату: `output_result(int x, double y)`- функція виведення результату розрахунків на екран.

Ці модулі дозволяють виконувати усі операції згідно з вимогами та реалізують логіку програми.

На підставі лістингу програми і виведеного результату можна сформулювати наступні обґрунтовані висновки для задачі 9.1:

✓ *Логічна структура програми:* програма має чітко структурований код з розділенням на окремі модулі для введення даних, розрахунку бонусів, конвертації бонусів у знижку та виведення результату. Це дозволяє забезпечити зручне управління програмою та полегшує її подальше розширення та обслуговування.

✓ *Правильність розрахунків*: програма правильно розраховує кількість нарахованих бонусів відповідно до встановлених правил. Також вона виконує конвертацію бонусів у знижку згідно з визначеною формулою. Це дозволяє користувачам отримувати точні та правильні результати після виконання програми.

✓ *Інтерфейс користувача*: програма має простий та зрозумілий інтерфейс для користувача, що дозволяє легко взаємодіяти з нею. Вона чітко виводить інструкції для користувача та результат розрахунків, що полегшує користувачеві розуміння та використання програми.

✓ *Тестування та стабільність:* Під час тестування програма виявила стабільну та надійну роботу, не виявлено жодних критичних помилок або непередбачених ситуацій. Це свідчить про те, що програма відповідає вимогам та може успішно використовуватися в реальних умовах.

Підсумовуючи виконану задачі 9.1, можна стверджувати:

- розроблена програма ефективно виконує вимоги до розрахунку бонусів та знижки для покупок у супермаркеті;

- модульна архітектура дозволяє легко розширювати та модифікувати програму у майбутньому;

- тестування підтвердило коректну роботу програми для різних вхідних даних;

- документація забезпечує зрозумілість коду та сприяє подальшому розвитку програми;

Наступним етапом було вирішення задачі 9.2, яка передбачала конвертацію температури з шкали Фаренгейта до шкали Цельсія. Виконавши аналіз вимог, написано функцію для конвертації температури з градусів Фаренгейта до градусів Цельсія. У реалізації була використана формула перетворення, яка враховує співвідношення між градусами Фаренгейта та Цельсія. Був реалізований тестовий драйвер для перевірки правильності роботи функції конвертації. Здійснене модульне тестування дозволило переконатися в правильності роботи функції та виявити можливі помилки.

У задачі 9.2 ортриманні такі артефакти:

1) функція `degree_conversion`, яка виконує конвертацію градусів з Фаренгейта до Цельсія.

2) функція `degree_fahrenheit`, яка призначена для введення кількості градусів за шкалою Фаренгейта.

3) функція `degrees_celsius`, яка виводить результат конвертації градусів до шкали Цельсія.

4) функція `task_9_2`, яка викликається для виконання задачі 9.2.

Ці артефакти включають в себе функції, які отримують вхідні дані, обробляють їх і повертають результати, що дозволяє виконати конвертацію градусів з Фаренгейта до Цельсія в рамках задачі 9.2.

На підставі лістингу програми і виведеного результату можна сформулювати наступні обґрунтовані висновки для цієї задачі:

✓ *Логічна структура програми*: програма має чітку логічну структуру, де функція конвертації `degree_conversion` відповідає за перетворення температури з градусів Фаренгейта до градусів Цельсія. Це дозволяє легко розуміти, як програма виконує свою основну функцію і спрощує процес розробки та управління кодом.

✓ *Правильність розрахунків*: перевірка результатів програми під час тестування підтвердила правильність розрахунків. Функція конвертації коректно виконує перетворення температури згідно з формулою, що гарантує надійність та точність обчислень.

✓ *Інтерфейс користувача*: інтерфейс користувача простий та зрозумілий. Користувачеві пропонується ввести температуру у градусах Фаренгейта, після чого він отримує відповідний результат у градусах Цельсія. Це забезпечує зручність використання програми та підвищує її придатність для широкого кола користувачів.

✓ *Тестування та стабільність*: результати модульного тестування свідчать про стабільність роботи програми. Тестовий драйвер успішно перевірив правильність роботи функції конвертації температури, а отже, програма може бути використана в реальних умовах без обурення стабільності чи точності результатів.

Підсумовуючи виконану задачі 9.2, можна стверджувати:

- розроблена програма ефективно виконує завдання з конвертації температури з градусів Фаренгейта до градусів Цельсія, відповідно до вказаних вимог;

- модульна архітектура дозволяє легко розширювати та модифікувати програму у майбутньому, наприклад, додавати підтримку інших шкал температури або розширювати функціональні можливості;

- тестування підтвердило коректну роботу програми для різних вхідних даних, що свідчить про надійність та стабільність її роботи;

- документація забезпечує зрозумілість коду та сприяє подальшому розвитку програми, дозволяючи легко розуміти її структуру та функціонал.

Далі, наступним етапом було розв'язання задачі 9.3, яка передбачала створення програми для обробки натурального числа N у визначеному діапазоні та повернення кількості двійкових нулів або одиниць в залежності від біта D числа N. Для підрахунку кількості бінарних 0 або 1 рекомендувалося використовувати тернарний оператор «?:». Після уважного аналізу вимог було визначено, що необхідно створити програму для обробки натурального числа N у вказаному діапазоні та повернення кількості двійкових нулів або одиниць в залежності від значення біта D числа. Була розроблена архітектура програми, яка включала функції для визначення біта D числа, обробки значення біта та підрахунку кількості двійкових нулів або одиниць. На основі цього проектування було реалізовано програмний модуль, який виконує зазначені завдання. Після втілення програмного модуля було проведено тестування для перевірки його коректності та відповідності вимогам. Були створені тестові набори для різних значень вхідних даних. Після успішного пройдення тестування програмний модуль був документований для забезпечення зрозумілості коду та спрощення його подальшого використання.

На підставі лістингу програми для задачі 9.3 можна виділити наступні артефакти:

1) функція `calculate_number`, яка призначена для підрахунку кількості бінарних 0 або 1 у числі N.

2) функція `natural_number`, яка використовується для введення натурального числа N.

3) функція `output_result`, яка виводить результат підрахунку кількості бінарних 0 або 1 у числі N.

4) функція `last`, яка використовується для навігації між різними завданнями, включаючи задачу 9.3.

5) функція `task_9_3`, яка викликається для виконання саме задачі 9.3.

Ці артефакти дозволяють користувачеві ввести натуральне число, обробити його та вивести результат, відповідно до вимог задачі 9.3.

На підставі лістингу програми і виведеного результату можна сформулювати наступні обґрунтовані висновки для задачі 9.3:

✓ *Логічна структура програми*: забезпечує послідовне виконання операцій. Спочатку користувач вводить натуральне число, потім програма обробляє це число та виводить результат. Кожна функція відповідає за свою частину завдання, що сприяє зрозумілості та легкості розвитку програми.

✓ *Розрахунки в програмі*: відповідають вимогам задачі, які полягають у підрахунку кількості бінарних 0 або 1 у введеному числі. Використання тернарного оператора у функції `calculate_number` дозволяє здійснити розрахунок ефективно та коректно.

✓ *Інтерфейс користувача*: простий та зрозумілий. Користувачеві пропонується ввести натуральне число, після чого він отримує результат обробки числа програмою. Тексти підказок допомагають користувачеві зрозуміти, як користуватися програмою.

✓ *Тестування та стабільність*: програма пройшла тестування для різних значень вхідних даних та продемонструвала стабільну та коректну роботу. Результати тестів підтвердили правильність розрахунків та коректність виведення результату.

Підсумовуючи виконану задачі 9.3, можна стверджувати:

- розроблена програма ефективно виконує завдання з обробки натурального числа та визначення кількості двійкових нулів або одиниць в залежності від значення біта D числа, відповідно до вказаних вимог;

- модульна структура програми дозволяє легко розширювати та модифікувати її функціонал у майбутньому, наприклад, додавати підтримку інших операцій або оптимізувати алгоритми;

- проведене тестування підтвердило коректну роботу програми для різних вхідних даних, що свідчить про надійність та стабільність її роботи;

- документація забезпечує зрозумілість коду та сприяє подальшому розвитку програми, дозволяючи легко розібратися у її структурі та функціоналі.

Завершенням роботи було вирішення задачі 9.4. у якій реалізовано програму, яка забезпечує інтерактивний інтерфейс для користувача, де введені символи

керують викликом конкретних функцій. Ключовим аспектом є можливість виклику різних функцій програми шляхом введення певних символів, що дозволяє зручно та ефективно керувати програмою за допомогою коротких команд. Аналізуючи хід виконання цієї задачі, спочатку був проведений аналіз вимог до програмного забезпечення. Задача передбачала створення програми, яка реагує на введені користувачем символи та викликає відповідні функції в залежності від введеного символу. Після аналізу вимог було розроблено архітектуру програми. Програма мала структуру, що дозволяє ефективно обробляти введені дані та вибирати потрібну функцію для виконання в залежності від введеного символу. Архітектура була розгорнута з урахуванням можливості подальшого розширення функціоналу програми. Детальне проектування програмного забезпечення включало реалізацію функцій, взаємодію з користувачем через консольний інтерфейс, а також обробку введених даних. Був розроблений тест-сьют для системного тестування програми. Тест-кейси були створені з метою перевірки коректності роботи програми в різних сценаріях введення символів користувачем. Тестування проводилося для перевірки правильності реакції програми на різні введені символи та відповідність очікуваному результату. В результаті аналізу та виконання всіх кроків було розроблено програмне забезпечення, яке відповідає вимогам та виконує потрібні функції відповідно до введених символів користувача. Тестування дало позитивні результати, підтверджуючи коректну роботу програми.

У задачі 9.4 ортриманні такі артефакти:

1) функція `last()`: ця функція є головним модулем керування виконанням програми. Вона виводить повідомлення для введення символу користувачем і викликає відповідні функції в залежності від введеного символу ('j', 'z', 'x', 'c', 'v', 'V', 'A'), а також обробляє помилкові введення.

2) функції `task_9_1()`, `task_9_2()`, `task_9_3()`, `task_8_1()`: ці функції виконують відповідні завдання, які вибираються користувачем під час виконання програми.

3) функція `read_data()`: запитує в користувача ввести суму покупки в гривнях та повертає це значення.

4) функція `output_result(int x, double y)`: виводить результат обчислення кількості нарахованих бонусів та суми до сплати з урахуванням знижки у гривнях.

5) функція `number_bonus(int x)`: обчислює кількість нарахованих бонусів в залежності від суми покупки x.

6) функція `discount_bonuses(int bonus)`: обчислює суму знижки в гривнях на основі кількості нарахованих бонусів.

7) функція `degree_fahrenheit()`: запитує в користувача кількість градусів Фаренгейта та повертає це значення.

8) функція `degrees_celsius(double f)`: виводить градуси у шкалі Цельсія на основі введених градусів Фаренгейта.

9) функція `calculate_number(unsigned short n)`: підраховує кількість бінарних 0 або 1 у числі n.

10) функція `natural_number()`: запитує в користувача число від 0 до 51950 та повертає це значення.

11) функція `output_result(unsigned short n)`: виводить результат підрахунку кількості бінарних 0 або 1 у введеному числі.

Ці артефакти відображають функціональність та можливості програми, що включає введення, обробку та виведення даних згідно з вказаними вимогами, а також допомагають у виконанні вимог до цієї задачі.

На підставі лістингу програми і виведеного результату можна сформулювати наступні обґрунтовані висновки для задачі 9.4:

✓ *Логічна структура програми:* програма має чітко структурований код, який дозволяє легко розуміти послідовність виконання операцій. Кожна функція відповідає за конкретне завдання: зчитування даних від користувача, обчислення результатів, виведення результатів тощо. Використання вказівників на функції в функції `last()` дозволяє динамічно викликати потрібну функцію залежно від введеного символу користувачем.

✓ *Правильність розрахунків*: розрахунки, виконані в програмі, є правильними і відповідають вимогам завдання.

✓ *Інтерфейс користувача:* інтерфейс користувача є зрозумілим і зручним для використання. Програма надає чіткі інструкції щодо введення символів для виклику різних функцій, а також повідомляє про помилкове введення символів і видає звуковий сигнал про помилку. Всі вимоги користувача щодо виходу з програми також враховані і чітко виконані.

✓ *Тестування програми:* успішно проведено системне тестування програми, включаючи різні сценарії введення та обробки даних. Всі тест-кейси були виконані згідно з планом тестування.

✓ *Стабільність програми*: під час тестування програма продемонструвала стабільну роботу та відсутність критичних помилок або виключень. Вона працювала безперебійно та ефективно навіть при різних видах введення користувача.

Підсумовуючи виконану задачі 9.4, можна стверджувати:

- розроблена програма ефективно виконує поставлену задачу з обробки натурального числа та визначення кількості двійкових нулів або одиниць в залежності від значення біта D числа;

- програма має чітку модульну структуру, що дозволяє легко розширювати та модифікувати її функціонал у майбутньому. Це забезпечує зручне управління програмою та збереження її стабільності під час розвитку;

- проведене тестування підтвердило коректну роботу програми для різних вхідних даних. Програма працює стабільно та надійно, не викликаючи критичних помилок під час виконання;

- програма надає зручний інтерфейс для взаємодії з користувачем, що дозволяє з легкістю вводити необхідні дані та отримувати очікувані результати.

На підставі всіх результатів у лабораторній роботі можна зробити наступні висновки:

1) виконані завдання дозволили закріпити та поглибити знання з програмування та роботи з Git;

2) отриманні навички проектування програмних модулів та їхньої реалізації сприятимуть подальшому розвитку як професійних, так і академічних навичок;

3) здійснене модульне тестування дозволить виявити та виправити можливі помилки в реалізації функцій, забезпечуючи високу якість програмного продукту.

Особисті враження від процесу виконання завдань лабораторної роботи були позитивними. Робота над задачами 9.1. - 9.4 дозволила глибше ознайомитися з мовою програмування С++, розібратися у принципах модульної розробки програмного забезпечення та відточити навички проектування та реалізації програмних рішень. Працюючи над завданнями, я отримала багато корисного досвіду у розв'язанні завдань з обробки даних, розробки логічних структур програм та взаємодії з користувачем. Виконання лабораторної роботи також дозволило краще розібратися з процесом тестування програмного забезпечення та важливістю забезпечення стабільності та надійності розроблених програмних продуктів. В цілому, виконання завдань лабораторної роботи було пізнавальним та цікавим, і дозволило поглибити мої знання та навички у програмуванні. Програма має потенціал для подальшого розширення та вдосконалення, наприклад, шляхом додавання нових функцій або оптимізації існуючого функціоналу.

## ВІДПОВІДЬ НА ЗАПИТАННЯ І ЗАВДАННЯ ДЛЯ САМОКОНТРОЛЮ ПІДГОТОВЛЕНОСТІ ДО ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ

1. Операції інкремента (++) та декремента (--) призначені для збільшення (інкремент) або зменшення (декремент) значення змінної на 1. Префіксна форма (++x, --x) змінює значення змінної перед її використанням, тоді як постфіксна форма (x++, x--) спочатку використовує поточне значення і потім змінює його.

2. `mch при char mch = 'D'; -mch;` матиме значення `'C'`. Отриманий результат полягає в тому, що ми використовуємо декрементний оператор `--` перед змінною `mch`, тому він зменшує значення змінної `mch` на 1. У вихідному стані `mch` має

значення `'D'`, яке відповідає десятковому числу 68 в ASCII коді. Після виконання `--mch;` ми отримуємо `'C'` (десяткове значення 67 в ASCII коді), оскільки ми зменшили значення змінної `mch` на 1.

3. Призначення складових операторів присвоювання в мовах програмування C/C++ полягає в зміні значення змінних за допомогою арифметичних або логічних операцій. Синтаксис запису цих операторів такий:

`=` - присвоює значення правої частини виразу змінній або константі.

`+=` - додає значення правої частини до значення змінної і присвоює результат змінній.

`-=` - віднімає значення правої частини від значення змінної і присвоює результат змінній.

`*=` - помножує значення змінної на значення правої частини і присвоює результат змінній.

`/=` - ділить значення змінної на значення правої частини і присвоює результат змінній.

`%=` - ділить значення змінної на значення правої частини і присвоює залишок від ділення результату змінній.

4. Оператори логічних операцій в мовах програмування C/C++ включають:

- Логічне І (AND): `&&`

*Операнди*: два булеві вирази або значення.

*Результат*: Логічне І поверне true (1), якщо обидва операнди мають значення `true`, і `false` (0) у всіх інших випадках.

- Логічне АБО (OR): `||`

*Операнди*: два булеві вирази або значення.

*Результат:* Логічне АБО поверне true (1), якщо хоча б один з операндів має значення `true`, і `false` (0) у всіх інших випадках.

- Логічне НЕ (NOT): `!`

*Операнд*: один булевий вираз або значення.

*Результат*: Логічне НЕ змінює значення операнду на протилежне. Якщо операнд має значення `true`, результат буде `false`, і навпаки.

5. Значення буде `false` в `comp при bool comp = (!0 && 1) == 0;` Оскільки `!0` - це "не 0", що дорівнює `true. true && 1` - обидва операнди істинні, тому результат - `true. true == 0` - порівняння істини з нулем, що дорівнює `false`. Таким чином, вираз `(!0 && 1) == 0` перетворюється на `true && 0`, що є `false`.

6. Оператори порозрядних операцій в мовах програмування C/C++ дозволяють виконувати операції на окремих бітах в бітових представленнях чисел. Ось перелік основних операторів порозрядних операцій, типів їх операндів і результатів виконання:

`&` (побітове І): *Типи операндів*: цілі числа. *Результат*: кожен біт результату є І-логічною кон'юнкцією відповідних бітів операндів.

`|` (побітове АБО): *Типи операндів*: цілі числа. *Результат*: кожен біт результату є АБО-логічною диз'юнкцією відповідних бітів операндів.

`^` (побітове виключне АБО): *Типи операндів*: цілі числа. *Результат*: кожен біт результату є виключним АБО відповідних бітів операндів.

`~` (побітове заперечення): *Типи операндів*: цілі числа. *Результат:* відображає кожен біт операнду в протилежне значення.

`<<` (зсув вліво): *Типи операндів*: цілі числа (лівий операнд), ціле число або беззнакове ціле число (правий операнд). *Результат*: лівий операнд зсувається вліво на кількість позицій, вказаних правим операндом.

`>>` (зсув вправо): *Типи операндів*: цілі числа (лівий операнд), ціле число або беззнакове ціле число (правий операнд). *Результат*: лівий операнд зсувається вправо на кількість позицій, вказаних правим операндом.

7. Операції виразів в мові програмування C/C++ мають різний пріоритет виконання. Відповідно до цього пріоритету, їх можна розташувати в порядку спадання:

1) Логічні операції (наприклад, `&&, ||, !`).

2) Порівняння (наприклад, `==`, `!=`, `<`, `>`, `<=`, `>=`).

3) Арифметичні операції (наприклад, `+`, `-`, `*`, `/`, `%`).

4) Інкремент/декремент (наприклад, `++`, `–`).

5) Складові операції присвоювання (наприклад, `=`, `+=`, `-=` і так далі).

8. Асоціативність операторів в мові програмування вказує на те, в якому порядку виконуються операції, коли вони мають однаковий пріоритет. Якщо є два оператори з однаковим пріоритетом (наприклад, додавання і віднімання), то асоціативність визначає, який з них буде виконуватися першим при відсутності додаткових директив. Наприклад, оператори додавання і віднімання зазвичай мають ліву асоціативність, що означає, що вони виконуються зліва направо: `a + b - c` обчислюється як `(a + b) - c`.

9. Значення `n` при: `int n = 0; n = !n<<1&1;` буде 0. Пояснення: Початкове значення `n` дорівнює 0. Вираз `!n` поверне протилежне значення `n`, тобто `!0` буде `true`, або `1`. Після цього вираз `!n << 1` виконає операцію зсуву вліво на один біт з результатом `1 << 1`, що дорівнює `2`. Вираз `2 & 1` виконає побітову кон'юнкцію між `2 і 1`, результатом буде 0.

10. Повне розгалуження (`if-else`) в мовах програмування C і C++ використовуються для. виконання певного блоку коду, якщо задана умова істинна (`true`), інакше виконується інший блок коду. Синтаксис:

```
if (условие) {
    // Код, который будет выполнен, если условие истинно
} else {
    // Код, который будет выполнен, если условие ложно
}
```

Неповне розгалуження (тернарний оператор) використовується для вибору одного з двох варіантів значень в залежності від умови. Синтаксис:

```
вираз1 ? вираз2 : выраз3;
```

`вираз1` - це умова; `вираз2` - це значення, яке повертається, якщо умова істинна; `вираз3` - це значення, яке повертається, якщо умова хибна.

11. Тернарний оператор (також відомий як оператор умови) - це спеціальний оператор у мові програмування C/C++, який дозволяє обирати одне з двох значень на основі умови. Його алгоритм виконання можна описати наступним чином: 1) Визначаємо умову; 2) Якщо умова істинна, то виконуємо `вираз1`. Якщо умова хибна - виконуємо `вираз2`.

12. Відмінність між результатом виконання оператора `break` та `continue` в тому, що `break` припиняє виконання циклу, а `continue` переходить до наступної ітерації без виконання наступних інструкцій в поточній ітерації.

Випадки використання оператора `break`: 1) коли умова виконана, і подальше виконання циклу не потрібне, `break` дозволяє негайно завершити цикл. 2) В обробнику виключень `break` може бути використаний для виходу з циклу або виконання інших дій, якщо виникає певна ситуація. 3) У складних операціях або в алгоритмах, `break` може бути використаний для негайного припинення виконання.

Випадки використання оператора `continue`: 1) коли певна умова не потребує обробки в поточній ітерації циклу, `continue`: дозволяє пропустити обробку і перейти до наступної ітерації. 2) У випадку пошуку або фільтрації даних, `continue` може бути використаний для ігнорування певних елементів або операцій. 3) Коли певні умови можуть призвести до зайвого виконання коду у циклі, `continue` допомагає уникнути зайвого виконання коду.

13. Цикли у програмуванні призначені для повторення виконання певного блоку коду декілька разів, зазвичай на основі певної умови. Вони дозволяють ефективно виконувати однотипні операції без необхідності повторного написання коду. У мовах програмування C/C++ є декілька видів циклів:

цикл `for` - використовується для повторення блоку коду певну кількість разів на основі умови або розгалуження.

цикл `while` - використовується для повторення блоку коду, доки певна умова залишається істинною.

цикл `do-while` - аналогічний циклу `while`, проте блок коду виконується принаймні один раз, навіть якщо умова не виконується.

14. Оператор `switch` використовується для вибору виконання певного блоку коду на основі значення змінної або виразу. Алгоритм його виконання:

1) Вираховується вираз, який використовується для розгалуження.

2) Порівнюється значення виразу з кожним з випадків (case) в операторі `switch`.

3) Якщо значення виразу відповідає значенню якого-небудь випадку (case), виконуються вказані дії в цьому випадку.

4) Після виконання дій в поточному випадку може бути використаний оператор `break`, щоб припинити виконання оператора `switch` і вийти з нього.

5) Якщо `break` відсутній, виконання коду продовжиться з наступним випадком після поточного випадку.

6) Якщо значення виразу не відповідає жодному випадку (case), виконуються дії в випадку `default`, якщо такий випадок присутній.

7) Після виконання дій в випадку `default` може бути також використаний оператор `break`, але це необов'язково.

8) Виконання оператора `switch` завершується, коли виконується оператор `break` або коли завершується останній випадок, або коли виконується випадок `default`.

У операторі `switch` перемикаючий вираз може бути цілим числом або символом, а константні вирази в кожному `case` можуть бути цілими числами або символами.

15. Синтаксис запису циклів `for`

```
for (ініціалізація; умова; крок) {
    // код, який виконується у циклі
}
```

Де `ініціалізація` - виконується один раз на початку циклу і використовується для ініціалізації змінних або виразів. `умова` - перевіряється перед кожним проходженням циклу. Якщо умова істинна, то виконання циклу

продовжується; якщо ні, цикл завершується. `крок` - виконується після кожного проходження циклу і використовується для зміни значень змінних, що контролюють цикл.

Синтаксис запису циклів `while`

```
while (умова) {
    // код, який виконується у циклі
}
```

Де `умова` - перевіряється перед кожним проходженням циклу. Якщо умова істинна, то виконання циклу продовжується; якщо ні, цикл завершується.

Синтаксис запису циклів `do…while`

```
do {
    // код, який виконується у циклі
} while (умова);
```

Де `умова` - перевіряється після кожного проходження циклу. Цикл виконується принаймні один раз, навіть якщо умова вже відразу є хибною.

Алгоритми виконання циклів:

циклів `for` : 1) Виконати ініціалізацію лічильника циклу. 2) Перевірити умову циклу. 3) Якщо умова виконується, виконати тіло циклу. 4) Виконати крок лічильника циклу. 5) Повторити кроки 2-4, поки умова циклу виконується.

циклів `while`: 1) Перевірити умову циклу. 2) Якщо умова виконується, виконати тіло циклу. 3) Повторити кроки 1-2, поки умова циклу виконується.

циклів `do…while`: 1) Виконати тіло циклу. 2) Перевірити умову циклу. 3) Якщо умова виконується, повторити крок 1.

16. Кваліфікатор типу `const` у мовах програмування C/C++ використовується для обмеження можливості зміни значення змінної після її ініціалізації.

Випадки доцільності застосування: 1) *оголошення констант:* коли потрібно оголосити значення, яке не буде змінюватися, наприклад, математичні константи або розміри масивів; 2) *параметри функцій:* коли параметр функції не має змінюватися всередині функції; 3) *покажчики:* коли потрібно забезпечити незмінність об'єктів через покажчики; 4) *класи і структури:* коли потрібно забезпечити незмінність членів класу або структури; 5) *ітератори:* коли ітератор має бути постійним і не має змінюватися під час ітерації.

Приклад, оголошення констант:

```
#include <iostream>

int main() {
    const int MAX_VALUE = 100;
    std::cout << "Max value: " << MAX_VALUE << std::endl;
    // Ми не можемо змінити значення константи MAX_VALUE
    // MAX_VALUE = 200; // Помилка компіляції
    return 0;
}
```

У цьому прикладі `MAX_VALUE` оголошено як константа за допомогою ключового слова `const`. Це значення не можна змінити після ініціалізації.

17. У мовах програмування C/C++ обов'язковим є тільки вираз умови в операторі `for`. Решта виразів (ініціалізація і модифікація) є необов’язковими. Оскільки вираз умова визначає, коли цикл має продовжувати своє виконання або коли він має завершити роботу. Це обов'язковий елемент, оскільки без умови немає способу визначити, коли потрібно завершити цикл. Це дозволяє забезпечити коректну роботу програми і уникнути безкінечного циклу. Ініціалізація виконується один раз перед входженням у цикл і використовується для початкового значення змінних. Вона не є обов'язковою частиною, оскільки можна ініціалізувати змінні поза циклом перед його початком. Модифікація виконується після кожної ітерації циклу і використовується для зміни значень змінних, що контролюють цикл. Це також необов'язковий елемент, оскільки можна змінювати значення змінних у тілі циклу.

18. В об’єкті `cout` при: `short int b = 0; cout << hex << ( ~b&0x80 );` виводиться шістнадцяткове число, оскільки використовано маніпулятор `hex`

19. Алгоритм перетворення функції C/C++ у функцію статичної бібліотеки C/C++:

1) Написати функцію або набір функцій у мові програмування C або C++.

2) Визначити прототип(и) функції у заголовочному файлі `.h` для C або `.hpp` для C++.

3) Реалізувати визначення функцій у файлі `.c` або `.cpp`.

4) Скомпілювати файл з кодом у статичну бібліотеку. Використати команди компіляції, такі як `gcc` або `g++` для цього.

5) Для використання функцій із статичної бібліотеки підключити відповідний заголовочний файл та скомпілюйте вашу програму із посиланням на статичну бібліотеку.

6) Запустити вашу програму, яка використовує функції з статичної бібліотеки.

Під час реалізації програмного забезпечення статичні бібліотеки використовуються для організації і структуризації коду, а також для підтримки його перевикористання.

20. Опис константи у заголовковому файлі використовується для централізованого визначення констант, які можуть бути використані у багатьох частинах програми або навіть у різних програмах. Приклад опису константи у заголовковому файлі C/C++ (назва файлу: `constants.h)`:

```
 #ifndef CONSTANTS_H
#define CONSTANTS_H

// Оголошення константи PI
const double PI = 3.14159265359;

#endif
```

## ВІДПОВІДЬ НА КОНТРОЛЬНІ ЗАПИТАННЯ І ЗАВДАННЯ

1. Тернарний оператор можназамінити повноцінним розгалуженням за допомогою оператора `if-else`.

Експериментальне доведення, задача: Дано змінну a. Якщо a більше 10, присвоїти змінній b значення a * 2, інакше a / 2.

Використання тернарного оператора в мові C++:

```
int a = 5;
int b = a > 10 ? a * 2 : a / 2; // b = 2
```

Використання if-else в мові C++:

```
int a = 5;
int b;
if (a > 10) {
    b = a * 2;
```

```
} else {
    b = a / 2;
}
// b = 2
```

Обидва фрагменти коду ефективно виконують одну й ту саму логіку: вони перевіряють, чи є a більшим за 10, і на основі цього визначають значення b. Це демонструє, що конструкція `if-else` може слугувати повноцінною заміною тернарному оператору, пропонуючи при цьому додаткову гнучкість та зручність для складніших логічних конструкцій.

2. Пріоритет виконання операцій і асоціативність — це основні поняття в програмуванні та математиці, які визначають порядок, в якому виконуються оператори в виразах.

Пріоритет виконання операцій вказує на послідовність, за якою виконуються різні типи операторів в математичних та логічних виразах. Оператори з вищим пріоритетом обчислюються перед операторами з нижчим пріоритетом. Асоціативність визначає порядок, в якому оператори з однаковим пріоритетом будуть виконуватися в виразі. Вона може бути лівоасоціативною або правоасоціативною.

3. Змінні, оголошені в тілі циклу або умови (вибору), мають локальну область видимості до блоку, в якому вони були оголошені. Це означає, що такі змінні доступні та можуть використовуватися лише в межах цього конкретного блоку коду (`for, while, if, switch,` і т.д.) та не доступні поза його межами.

Експериментальне доведення. Розглянемо наступний код на мові C++ як приклад:

```
 #include <iostream>

int main() {
    if (true) {
        int x = 5; // Змінна x оголошена в блоку if
        std::cout << "Inside if: x = " << x << std::endl;
    }
    // Спроба використати змінну x поза блоком if
     // std::cout << "Outside if: x = " << x << std::endl; // Цей рядок спричинить помилку компіляції

    for (int i = 0; i < 1; i++) {
        int y = 10; // Змінна y оголошена в блоку for
        std::cout << "Inside for: y = " << y << std::endl;
```

```
    }
    // Спроба використати змінну y поза блоком for
      // std::cout << "Outside for: y = " << y << std::endl; // Цей рядок також спричинить помилку компіляції

    return 0;
}
```

У цьому прикладі, змінні x та y оголошені в межах блоків `if` та `for` відповідно. Якщо спробувати скомпілювати цей код із розкоментованими рядками, які спробують вивести значення x та y за межами їх блоків, компілятор видасть помилку, скаржачись на те, що x та y не були оголошені в цій області. Це демонструє, що область видимості змінних, оголошених в блоках, обмежена цими самими блоками.

4. Асоціативність операторів визначає, як оператори того самого пріоритету групуються в умовах відсутності дужок. Ось огляд асоціативності для різних типів операторів в мовах програмування C та C++. Нижче наведено асоціативність різних типів операторів у мові C/C++:

*Арифметичні операції:*

Бінарні *, /, % (множення, ділення, залишок від ділення): лівоасоціативні.

Бінарні +, - (додавання, віднімання): лівоасоціативні.

Унарні +, - (універсальний плюс, унарний мінус): правоасоціативні.

*Логічні операції:*

&& (логічне І): лівоасоціативний.

|| (логічне АБО): лівоасоціативний.

! (логічне НЕ): правоасоціативний.

*Логічні порозрядні операції:*

&, |, ^ (порозрядне І, порозрядне АБО, порозрядне виключне АБО): лівоасоціативні.

~ (порозрядне НЕ): правоасоціативний.

<<, >> (зсув вліво, зсув вправо): лівоасоціативні.

*Операції інкремента та декремента:*

Префіксні ++, -- (інкремент, декремент): правоасоціативні.

Постфіксні ++, -- (інкремент, декремент): лівоасоціативні.

*Тернарна операція:*

?: (тернарний умовний оператор): правоасоціативний.

*Операції порівняння:*

==, != (рівність, нерівність): лівоасоціативні.

<, <=, >, >= (менше, менше або рівне, більше, більше або рівне): лівоасоціативні.

5. Тернарний оператор `?`: у C/C++ є корисним для здійснення коротких умовних присвоєнь або виборів без необхідності використовувати більш об'ємні конструкції `if-else`. Він особливо доцільний у наступних випадках:

1) Просте умовне присвоєння: коли потрібно присвоїти змінній одне з двох значень на основі якоїсь умови (C++).

```
int a = 10, b = 5;
int max = a > b ? a : b; // Якщо a > b, присвоїти a змінній max, інакше b.
```

2) Тернарний оператор може бути вбудованим у складніші вирази для здійснення умовного вибору значення (C++).

```
int score = 75;
std::cout << "Your grade is " << (score > 60 ? "passed" : "failed") << std::endl;
```

3) Умовне виконання функцій: коли потрібно вибрати, яку функцію викликати, залежно від умови (C++).

```
int x = 10, y = 20;
std::cout << (x > y ? max(x, y) : min(x, y)) << std::endl; // Викликає max, якщо x
> y, інакше min.
```

4) Тернарний оператор можна використовувати для умовної ініціалізації змінних при їх оголошенні (C++)

```
bool isAdult = true;
std::string status = isAdult ? "Adult" : "Minor"; //
```

6. Після виконання інструкції `cnt--;`, значення змінної `cnt` буде зменшено на 1. Це є скороченим записом для зменшення значення змінної на 1. Це можна записати як `cnt = cnt - 1;`, але використання оператора декремента `--` дозволяє скоротити код.

7. Константна змінна, оголошена за допомогою кваліфікатора типу `const`, відрізняється від звичайної змінної наступними особливостями: 1) *Незмінність значення:* значення константної змінної не може бути змінено після її ініціалізації. Якщо спробувати змінити значення константної змінної, компілятор видасть

помилку. 2) *Обов'язкова ініціалізація:* константну змінну потрібно ініціалізувати в момент її оголошення. Неініціалізована константна змінна призводить до помилки компіляції. 3) *Простір імен:* константні змінні можуть бути оголошені у глобальному просторі імен або у просторі імен функції, а також у класах. Вони можуть бути оголошені зовні або всередині функцій або класів. 4) *Компіляторна оптимізація: к*онстантні змінні можуть бути піддані компіляторній оптимізації, що дозволяє компілятору виконувати ряд оптимізацій, зокрема заміну константних виразів на їхні значення під час компіляції.

Змінну варто оголошувати в таких випадках: 1) коли значення змінної не має змінюватися протягом усього часу виконання програми або після її ініціалізації; 2) щоб запобігти випадковій змінні значення змінної в коді. Це допомагає уникнути помилок, пов’язаних з неправомірним переприсвоєнням значень; 3) оптимізація компілятора; 4) читабельність і підтримка коду.

8. У мовах програмування C і C++, логічні оператори працюють з булевими значеннями (`true` або `false`). Операнди логічних операторів можуть бути будь-якого типу, який може бути конвертований в булевий тип. Ось деякі основні типи, які можуть використовуватися як операнди для логічних операторів в мовах C та C++:

1) Цілі числа (`int, long, short`): у C/C++ ненульове ціле число розглядається як `true`, а нульове - як `false`.

2) Знакові цілі числа (`signed`): так само, як і зі звичайними цілими числами.

3) Беззнакові цілі числа (unsigned): так само, як і зі звичайними цілими числами.

4) Логічні значення (`bool`): у мові C++ існує вбудований тип `bool`, який має значення `true` або `false`.

5) Вказівники (`pointers`): Вказівники перевіряються на нульове значення. Нульовий вказівник розглядається як `false`, а будь-який ненульовий вказівник - як `true`.

6) Вказівники на функції (`function pointers`): як і звичайні вказівники, нульовий вказівник на функцію розглядається як `false`, а ненульовий - як `true`.

Ці типи можуть бути використані як операнди в логічних виразах умовних перевірок (`if, while, for`, тощо) або в логічних операціях (`&&, ||, !`).

9. У виразі `bool cnt = !!0;`, використовується оператор "НЕ" (`!`). Значення `!!0` означає "не 0". Спочатку `!0` повертає `true`, оскільки нуль (0) розглядається як `false`, та операція "НЕ" його змінює на протилежне значення, тобто `true`.
Потім вираження `!true` знову повертає `false`, оскільки операція "НЕ" перетворює `true` на `false`. Отже, результатом виразу `!!0` буде `false`. Таким чином, змінна `cnt` буде мати значення `false`, оскільки вираз `!!0` дає `false`.

10. Правило запису виразу ініціалізації у циклах з параметром (`for`) в мові програмування C++ полягає в наступному: у виразі ініціалізації може бути один або декілька виразів, розділених комами. Ці вирази виконуються один раз перед початком циклу і використовуються для ініціалізації змінних, які будуть використовуватися в циклі, або для виконання інших дій, необхідних перед початком циклічного виконання.