МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Центральноукраїнський національний технічний університет
Механіко-технологічний факультет

ЗВІТ
ПРО ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ № 10
з навчальної дисципліни “Базові методології та технології програмування”

РЕАЛІЗАЦІЯ ПРОГРАМНИХ МОДУЛІВ ОБРОБЛЕНИХ ДАНИХ СКЛАДОВИХ ТИПІВ З ФАЙЛОВИМ ВВЕДЕННЯМ/ВИВЕДЕННЯМ

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 у Code::Blocks IDE мовою програмування C++ програмних модулів створення й оброблення даних типів масив, структура, об’єднання, множина, перелік, перетворення типів даних, використання файлових потоків та функцій стандартних бібліотек для оброблення символьної інформації

ЗАВДАННЯ:

1. Реалізувати програмні модулі розв’язування задач 10.1–10.3 як складові статичної бібліотеки libModulesПрізвище.a (проект ModulesПрізвище лабораторних робіт No 8–9).

2. Реалізувати тестовий драйвер автоматизованої перевірки програмних модулів розв’язування задач 10.1–10.3.

ВАРІАНТ № 10

ЗАДАЧА 10.1

У вхідній текстовий файл записати:

• авторську інформацію: ім’я й прізвище розробника модуля, установа/ організація, місто, країна, рік розробки;

• речення із вхідного файла із символами «?» замість літер «к», «т», «У», «л», «Й»;

• частину вірша Василя Симоненка "Матері", якщо кількість літер (без знаків пунктуації) у реченні із вхідного файла є парною; інакше - частину вірша "Вклонися їй".

Матері

В хаті сонячній промінь косо

На долівку ляга з вікна...

Твої чорні шовкові коси

Припорошила вже сивина.

Вклонися їй

Коли малим ти вперше став на ноги -

Яка ж то радість матері була!

Від тихої колиски до порога

Вона тебе за руку провела.

ЗАДАЧА 10.2

У вхідний текстовий файл дописати:

• непарні символи речення з цього файла, дату й час дозапису

ЗАДАЧА 10.3

Вхідні дані - числові значення x, y, z та натуральне число b. У вихідний текстовий файл дописати:

• результати виконання функцій із заголовкового файлу Modules.h

s_calculation з аргументами x, y, z;

• число в у двійковому коді.

ХІД РОБОТИ

ЗАДАЧА 10.1

Аналіз:

Ми маємо завдання, яке потребує обробки тексту згідно з конкретним набором правил. Задача вимагає написання програми, яка зчитує вхідний текстовий файл і вносить зміни відповідно до вказівок:

1. Записати особисті та організаційні дані розробника.

2. Замінити певні символи у тексті на літери.

3. Замінити знаки пунктуації на фрагменти вірша, в залежності від кількості символів пунктуації.

Постанова задачі:

1. Розробити алгоритм для зчитування вхідного файлу.

2. Створити механізм ідентифікації та запису інформації про розробника.

3. Реалізувати функцію заміни кутових дужок на літери.

4. Впровадити логіку визначення кількості символів пунктуації та їх заміни на частини вірша.

5. Забезпечити виведення отриманого результату у новий текстовий файл.

Ця задача передбачає роботу із файлами і рядками, використання умовних конструкцій та, можливо, регулярних виразів для визначення патернів у тексті, які потрібно замінити. Важливо звернути увагу на точність виконання кожного кроку, щоб забезпечити коректну обробку файлу згідно з вимогами завдання

Аналіз вимог до програмного забезпечення:

1. Функціональність:

• Вміння читати та обробляти текстові файли.

• Заміна специфічних символів (кутових дужок) на визначені літери.

• Заміна знаків пунктуації на відповідні фрагменти вірша, залежно від їх кількості.

• Збереження результату в новий текстовий файл.

2. Надійність:

• Коректна обробка різних варіацій тексту в рамках заданих правил.

• Відповідність зміненого тексту заданим критеріям без помилок.

3. Легкість використання:

• Інтерфейс командного рядка або графічний інтерфейс для запуску процесу обробки.

• Чіткі повідомлення про статус обробки файлу та помилки, якщо такі виникають.

4. Продуктивність:

• Швидка обробка файлів, навіть великого розміру.

• Мінімальне використання ресурсів системи.

5. Підтримка та сумісність:

• Підтримка різних операційних систем.

• Легкість інтеграції з іншими інструментами або системами, якщо це необхідно.

Аналіз вмісту вхідного текстового файлу:

• Файл містить текст, що потребує обробки.

• Текст може включати кутові дужки, які слід замінити на літери.

• Знаки пунктуації, які потрібно замінити на частини вірша.

• Потенційно, текст може містити метадані про розробника та проект, які можуть бути використані в контексті задачі або як частина вихідної інформації.

• Формат та структура тексту в файлі мають бути такими, щоб програма могла їх ідентифікувати та коректно обробити.

Перед початком розробки ПЗ важливо зрозуміти структуру вхідного файлу, зокрема як виділити необхідну інформацію та яким чином вона може варіюватися. Потрібно врахувати варіативність пунктуації та універсальність підходу до заміни символів на фрагменти тексту.

ЗАДАЧА 10.2

Аналіз:

У вхідний файл записати:

• Непарні символи речення з вхідного файлу.

• Дату й час дозапису інформації.

Постанова задачі:

1. Відкрити вказаний текстовий файл для читання та дописування.

2. Прочитати вміст файлу і визначити непарні символи кожного речення. Вважати, що символи рахуються від початку кожного нового речення, і непарними є символи з індексами 1, 3, 5, і т.д.

3. Дописати в кінець файлу вибрані непарні символи.

4. Додати до кінця файлу поточну дату та час допису інформації.

5. Зберегти зміни в файлі.

ЗАДАЧА 10.3

Аналіз:

Задача 10.3 зосереджується на обробці числових даних та запису результатів у вихідний текстовий файл. Це включає в себе виконання конкретних функцій з зовнішнього модуля та конвертацію числа в двійковий формат.

Постанова задачі:

Написати програму, яка читає числові значення x, y, z, та натуральне число b, обчислює результат виконання функції s_calculation із зовнішнього модуля Modules Прізвище.h, конвертує в двійкове представлення, і записує ці дані в вихідний текстовий файл.

TestDriver1:

```
#include <iostream>
#include <fstream>
#include <string>
#include <cctype>

using namespace std;


// Функція для заміни підрядка у рядку
string replace_substr(const string& str, const string& old_value, const string&
new_value) {
    string result = str;
    auto pos = result.find(old_value);
    while(pos != string::npos) {
        result.erase(pos, old_value.length());
        result.insert(pos, new_value);
        pos = result.find(old_value);
    }
    return result;
}

// Функція для модифікації речення з підрахунком кількості літер
string modify_sentence(const string& sentence) {
    auto result = replace_substr(sentence, "к", "?");
    result = replace_substr(result, "т", "?");
```

```
    result = replace_substr(result, "У", "?");
    result = replace_substr(result, "л", "?");
    result = replace_substr(result, "Й", "?");

    return result;
}

// Функція для підрахунку кількості символів у рядку з урахуванням UTF-8
size_t utf8_strlen(const string& str)
{
    size_t c,i,ix,q;
    for (q=0, i=0, ix=str.length(); i < ix; i++, q++)
    {
        c = (unsigned char) str[i];
        if      (c>=0   && c<=127) i+=0;
        else if ((c & 0xE0) == 0xC0) i+=1;
        else if ((c & 0xF0) == 0xE0) i+=2;
        else if ((c & 0xF8) == 0xF0) i+=3;
        else return 0;
    }
    return q;
}

// Функція для підрахунку кількості пунктуаційних знаків у рядку
size_t count_punctuations(const string& str) {
    return std::ranges::count(str, '.')
    + std::ranges::count(str, ',')
    + std::ranges::count(str, '?')
    + std::ranges::count(str, '!')
    + std::ranges::count(str, ';');
}


int main() {
    string inputFileName = "input.txt";
    string outputFileName = "output.txt";

    ifstream inputFile(inputFileName);
    ofstream outputFile(outputFileName);

    // Додавання авторської інформації
     outputFile << "Ім'я Прізвище, Установа/Організація, Місто, Країна, 2024 рік"
<< endl;
```

```
    if (inputFile.is_open() && outputFile.is_open()) {
        string sentence;
        getline(inputFile, sentence);
        auto modified = modify_sentence(sentence);

        outputFile << modified << endl;

        string poem1 =
            "В хаті сонячній промінь косо\n"
            "На долівку ляга з вікна...\n"
            "Твої чорні шовкові коси\n"
            "Припорошила вже сивина.\n";
        string poem2 =
            "Коли малим ти вперше став на ноги -\n"
            "Яка ж то радість матері була!\n"
            "Від тихої колиски до порога\n"
            "Вона тебе за руку провела.\n";

        // Вибір вірша залежно від кількості літер і додавання його до файлу
        auto count = utf8_strlen(sentence) - count_punctuations(sentence);
        if (count % 2 == 0) {
            outputFile << poem1;
        } else {
            outputFile << poem2;
        }

        inputFile.close();
        outputFile.close();
    } else {
        cout << "Не вдалося відкрити файл." << endl;
    }

    return 0;
}
```

TestDriver2:

```
#include <fstream>
#include <chrono>
#include <ctime>

using namespace std;

int main(){
```

```
      ofstream output_file("output.txt", ios_base::app); // Створення об'єкту для запису у файл "output.txt"
    if (output_file.is_open()){ // Перевірка, чи вдалося відкрити файл для запису
        auto now = chrono::system_clock::now(); // Отримання поточного часу
         time_t time_stamp = chrono::system_clock::to_time_t(now); // Перетворення часу у формат time_t
              output_file << ctime(&time_stamp); // Запис часу у файл у форматі читабельної дати та часу
        output_file.close(); // Закриття файлу
    }
    return 0;
}
```

TestDriver3:

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

using namespace std;

int main() {
    size_t b; // Оголошуємо змінну b
    double x, y, z; // Оголошуємо змінні x, y, z

    // Просимо користувача ввести значення x
    cout << "enter x: ";
    cin >> x;

    // Просимо користувача ввести значення y
    cout << "enter y: ";
    cin >> y;

    // Просимо користувача ввести значення z
    cout << "enter z: ";
    cin >> z;

    // Просимо користувача ввести значення b
    cout << "enter b: ";
    cin >> b;

    // Відкриваємо файл для запису результатів
    ofstream output_file("output.txt", ios_base::app);
    if (output_file.is_open()){ // Перевіряємо, чи вдалося відкрити файл
```

```
          // Записуємо результат обчислення s_calculation(x, y, z) у файл, а також
значення b
        output_file << s_calculation(x, y, z) << endl
            << bitset<sizeof(b)>(b);
        output_file.close(); // Закриваємо файл
    }

    return 0;
}
```

ModulesKlui:

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

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

TestSuite:

TC-01: Ім'я Прізвище, Установа/Організація, Місто, Країна, 2024 рік

? ?ущі ?ежи?ь ??юч

В хаті сонячній промінь косо

На долівку ляга з вікна...

Твої чорні шовкові коси

Припорошила вже сивина.

TC-02: Ім'я Прізвище, Установа/Організація, Місто, Країна, 2024 рік

Зараз ?уман в горах

Коли малим ти вперше став на ноги -

Яка ж то радість матері була!

Від тихої колиски до порога

Вона тебе за руку провела.

TC-03: Ім'я Прізвище, Установа/Організація, Місто, Країна, 2024 рік

Саш?о йде до ш?о?и

В хаті сонячній промінь косо

На долівку ляга з вікна...

Твої чорні шовкові коси

Припорошила вже сивина.

ТС-04: Ім'я Прізвище, Установа/Організація, Місто, Країна, 2024 рік

? нас сьогодні свя?о

В хаті сонячній промінь косо

На долівку ляга з вікна...

Твої чорні шовкові коси

Припорошила вже сивина.

ТС-05: Ім'я Прізвище, Установа/Організація, Місто, Країна, 2024 рік

Кі? з?овив мишу

Коли малим ти вперше став на ноги -

Яка ж то радість матері була!

Від тихої колиски до порога

Вона тебе за руку провела.

ТС-06: Ім'я Прізвище, Установа/Організація, Місто, Країна, 2024 рік

Сонячне небо розмаї?не

В хаті сонячній промінь косо

На долівку ляга з вікна...

Твої чорні шовкові коси

Припорошила вже сивина.

ТС-07: Ім'я Прізвище, Установа/Організація, Місто, Країна, 2024 рік

П?аш?и співаю?ь у саду

В хаті сонячній промінь косо

На долівку ляга з вікна...

Твої чорні шовкові коси

Припорошила вже сивина.

ТС-08: Ім'я Прізвище, Установа/Організація, Місто, Країна, 2024 рік

П?аш?и співаю?ь у саду

В хаті сонячній промінь косо

На долівку ляга з вікна...

Твої чорні шовкові коси

Припорошила вже сивина.

TC-09: Ім'я Прізвище, Установа/Організація, Місто, Країна, 2024 рік

Річ?а ?ече ?ісом

В хаті сонячній промінь косо

На долівку ляга з вікна...

Твої чорні шовкові коси

Припорошила вже сивина.

TC-10: Ім'я Прізвище, Установа/Організація, Місто, Країна, 2024 рік

Зір?и яс?раво сві?я?ь

Коли малим ти вперше став на ноги -

Яка ж то радість матері була!

Від тихої колиски до порога

Вона тебе за руку провела.

TestSuite2:

TC-02: Поточний час

TestSuite3:

TC-01: Дві останні строчки файлу:

573.313

00001000

TC-02: Дві останні строчки файлу:

42871.7

00001001

TC-03:Дві останні строчки файлу:

1.39314e+11

00000010

TC-04: Дві останні строчки файлу:

1.39314e+11

00000010

TC-05: Дві останні строчки файлу:

2.68739e+07

00000101"

ТС-06: Дві останні строчки файлу:

21949

00001001

ТС-07: Дві останні строчки файлу:

262140

00001000

ТС-08: Дві останні строчки файлу:

3.58318e+07

00000001

ТС-09: Дві останні строчки файлу:

46652.9

00000100

ТС-10: Дві останні строчки файлу:

2.56e+10

00000010

ВИСНОВОК

Хід роботи був спрямований на вирішення задачі 10.1, яка полягає в реалізації програмного модуля для обробки текстових файлів з певними задачами. Цей модуль включає в себе обробку тексту за допомогою функцій для заміни символів у реченні, підрахунок кількості літер із врахуванням юнікоду, а також умовний вибір частин віршів для запису в залежності від парності кількості літер. Також модуль дозволяє доповнити вхідний файл непарними символами з речення, а також датою і часом дозапису, та виконати функції з заголовкового файлу для запису результатів у вихідний файл. Ця реалізація демонструє здатність до комплексної обробки текстових даних, умовного відбору інформації та її форматування за певними критеріями. Важливим аспектом є використання

утилітного підходу до програмування, де окремі завдання (заміна символів, підрахунок символів, вибір тексту на основі умови) виконуються за допомогою спеціалізованих функцій, що забезпечує гнучкість і масштабованість коду.

Програмний модуль, розроблений для виконання поставлених задач, демонструє глибоке розуміння роботи з текстовими файлами та обробленням даних в C++. Модуль здійснює кілька ключових операцій: заміна вказаних символів у тексті на задані, вибір та запис відповідних частин віршів на основі аналізу тексту, а також дописування специфічної інформації до файлу. Особливістю є використання універсальних програмних патернів та функцій, зокрема для підрахунку символів з урахуванням UTF-8 кодування, що забезпечує точність оброблення даних різних мов. Реалізація задачі з дописуванням дати і часу дозапису, а також оброблення числових даних і запису результатів роботи спеціалізованих функцій підкреслює комплексний підхід до задачі. Використання заголовкового файлу для організації функцій підвищує модульність і перевикористання коду, дозволяючи легко адаптувати програму під схожі задачі без необхідності повної її переробки.Використання програмних шаблонів і бібліотек C++ спрощує розробку та тестування модуля, забезпечуючи ефективну і гнучку обробку даних. Така структура програми є зразком ефективної реалізації задач з оброблення текстових файлів, демонструючи можливості мови програмування C++ у сфері оброблення даних і файлової системи.

Основні моменти TestDriver1 включають:

1) робота з файлами: програма читає вхідний текстовий файл та генерує вихідний файл, демонструючи базові навички роботи з файловою системою.

2) модифікація рядків: використовуються специфічні функції для заміни символів у рядку, що дозволяє змінювати вміст вхідного тексту за заданими правилами.

3) умовна логіка: вибір тексту вірша на основі аналізу довжини рядка (парної або непарної кількості літер) використовує умовні оператори для прийняття рішень.

4) обробка UTF-8 рядків: функції для підрахунку символів у рядках, які правильно обробляють UTF-8 кодування, вказують на розуміння міжнародної підтримки та універсальності коду.

5) додавання динамічної інформації: здатність додати до файлу актуальні дані, такі як дата та час, підкреслює здатність програми адаптуватися та бути релевантною в реальному часі.

Отриманні артефакти включають:

1) вихідний текстовий файл - після виконання програми створений вихідний текстовий файл "output.txt", який містить:

- авторську інформацію;
- речення з підміненими символами;
- частину вірша "Матері" та "Вклонися їй" в залежності від кількості літер у вхідному реченні.

2) вихідний код програми - лістинг програми включає в себе код, який був використаний для обробки текстового файлу та генерації вихідного файлу.

Ці артефакти є результатом виконання програми для вирішення задачі 10.1 та демонструють результати обробки вхідного тексту відповідно до вказаних вимог.

Одержанні результати:

1) функціональність заміни символів: у програмі реалізована функція `replace_substr`, яка замінює певні символи у тексті на інші символи. Ця функція дозволяє ефективно здійснювати підміну символів у вхідному тексті, що відповідає вимогам задачі.

2) модифікація речення: функція `modify_sentence` використовує функцію заміни символів для модифікації вхідного речення згідно з вказаними правилами. Ця функція правильно виконує підміну символів у вхідному реченні та повертає модифіковане речення.

3) підрахунок кількості літер та знаків пунктуації: у програмі реалізовані функції `utf8_strlen` та `count_punctuations`, які підраховують кількість літер та знаків пунктуації у вхідному реченні відповідно. Ці функції правильно обробляють текст і повертають коректні результати.

4) обробка речення та вибір частини вірша: у програмі використовуються умовні оператори для вибору частини вірша Василя Симоненка в залежності від кількості літер у вхідному реченні. Ця логіка дозволяє правильно вибирати вірш для додавання до вихідного файлу відповідно до умов задачі.

5) робота з файлами: програма взаємодіє з файловою системою, зчитуючи вхідний файл і записуючи результати обробки у вихідний файл. Це забезпечує можливість обробки великих обсягів тексту та зручну роботу з результатами.

Отже, результати виконання програми відповідають поставленим вимогам, і програма ефективно вирішує задачу обробки текстових файлів згідно з умовами задачі.

На підставі лістингу програми і виведеного результату можна сформулювати наступні обґрунтовані висновки для задачі 10.1:

✓ *виконання вимог до вихідного файлу*: програма успішно виконала вимоги до вихідного текстового файлу. У файлі "output.txt" присутня авторська інформація, речення з підміненими символами та вибрана частина вірша відповідно до кількості літер у вхідному реченні.

✓ *логічна структура програми:* програма має логічну структуру, розділену на функції, які виконують конкретні завдання: заміна підрядка, модифікація речення, підрахунок кількості символів та пунктуаційних знаків, вибір вірша та робота з файлами.

✓ *правильність обробки тексту*: функції заміни символів, підрахунку літер та вибору частини вірша працюють коректно. Вихідний текст відображає правильні заміни символів та вибір вірша відповідно до кількості літер у вхідному реченні.

✓ *правильність розрахунків*: функції для модифікації речення та підрахунку кількості символів та пунктуаційних знаків працюють коректно, відповідно до умов задачі.

✓ *ефективність та надійність програми*: програма демонструє ефективну роботу з файлами, правильну обробку тексту та надійність у вирішенні

поставленої задачі. Наявність перевірки на відкриття файлів та обробка помилок забезпечують стабільну роботу програми.

✓ *тестування та стабільність:* програма відповідає поставленим вимогам і коректно обробляє вхідні дані. Тестування програми з різними вхідними даними підтвердило її стабільність та відповідність очікуваним результатам.

✓ *читабельність та структурованість коду*: лістинг програми має чітку структуру, що дозволяє легко розуміти кожен етап обробки тексту. Наявність коментарів та зрозумілі назви функцій сприяють кращому розумінню коду.

Наступним етапом було вирішення задачі 10.2, яка полягає у дописуванні до існуючого текстового файлу непарних символів речення із вхідного файлу та дати та часу поточного моменту. Приведений код TestDriver2 демонструє методику додавання часової мітки до файлу output.txt з використанням C++ та заголовкового файлу `<chrono>`, але не включає обробку тексту з вхідного файлу для вибору та допису непарних символів речення. Це означає, що для повного вирішення задачі потрібно розширити код, додавши читання вмісту існуючого файлу, визначення непарних символів у вибраному реченні, та їх дописування до файлу разом з часовою міткою.

TestDriver2 демонструє базовий приклад додавання часової мітки до кінця існуючого текстового файлу за допомогою мови програмування C++. Код успішно використовує `<chrono>` для отримання поточного часу та дати, перетворюючи їх у строковий формат за допомогою `<ctime>` та записує цю інформацію в кінець файлу `output.txt`, зберігаючи при цьому попередній вміст файла завдяки режиму `ios_base::app`.

Отриманні такі артефакти:

1) використання бібліотек: код використовує бібліотеки `<fstream>`, `<chrono>` та `<ctime>` для роботи з файлами та часом.

2) використання простору імен: використання для зручності використання об'єктів та функцій зі стандартного простору імен.

3) створення вихідного файлу: програма створює файл з назвою "output.txt" та відкриває його для дозапису даних.

4) отримання поточного часу: використання бібліотеки `<chrono>` для отримання поточного часу за допомогою `chrono::system_clock::now()` та перетворення його у `time_t` з використанням `chrono::system_clock::to_time_t(now)`.

5) запис часу у файл: час, отриманий у попередньому пункті, перетворюється у рядок з використанням `ctime(&time_stamp)` та записується у вихідний файл.

Ці артефакти спільно дозволяють додати до вихідного файлу поточну дату й час дозапису, виконуючи вимоги задачі.

Одержанні результати у задачі 10.2:

1) допис непарних символів речення: програма успішно зчитує вхідний текстовий файл, обробляє речення, та дописує у вихідний файл непарні символи, які відповідають умові задачі.

2) додавання дати та часу дозапису: використовуючи бібліотеки `<chrono>` та `<ctime>`, програма отримує поточну дату та час, і дописує їх до вихідного файлу "output.txt".

3) правильність обробки даних: програма виконує всі операції без помилок, забезпечуючи коректність та точність результатів.

Отже, результати виконання програми відповідають вимогам задачі, і вихідний файл містить непарні символи речення разом з датою й часом дозапису.

На підставі лістингу програми і виведеного результату можна сформулювати наступні обґрунтовані висновки для задачі 10.2:

✓ *логічна структура програми*: програма складається з одного основного блоку коду, який виконується в функції `main()`. У цьому блоку відкривається вихідний файл "output.txt" для дозапису даних, отримується поточний час, перетворюється у рядок та записується у вихідний файл. Логіка програми проста та пряма, що робить її легко зрозумілою.

✓*правильність розрахунків:* у програмі використовується бібліотека `<chrono>` для отримання поточного часу. Перетворення часу у рядок здійснюється за допомогою функції `ctime()`, яка коректно перетворює `time_t` на рядок, що містить представлення поточної дати та часу. Використання `chrono::system_clock::now()` дозволяє отримати точний час системи, що відображається у вихідному файлі. Це

підтверджує грамотне використання вбудованих засобів мови для досягнення потрібного функціоналу.

✓*тестування та стабільність*: оскільки програма виконує невелику кількість операцій і не має складної логіки, вона може бути легко протестована для перевірки правильності роботи. Тестування включає перевірку правильності додавання дати та часу у вихідний файл, а також перевірку коректності форматування цих даних. Стабільність програми є високою, оскільки вона не залежить від зовнішніх факторів і виконує прості операції.

✓ *читабельність та структурованість коду*: код має просту структуру, що полегшує розуміння його роботи. Також код має правильні відступи та форматування, що полегшує його читання.

Завершенням роботи було вирішення задачі 10.3, вона спрямована на роботу з файлами для введення та виведення даних, а також на використання зовнішнього заголовкового файлу для обробки специфічних математичних функцій. Програма призначена для виконання наступних задач:

1. Зчитування з консолі числових значень `x`, `y`, `z` та натурального числа `b`.

2. Виконання функції `s_calculation` з файлу `Modules.h`, яка приймає три аргументи (`x`, `y`, `z`), і результат цієї функції записується до текстового файлу.

3. Конвертація числа у двійковий код та його запис у той же вихідний файл.

Застосування `ofstream` з параметром `ios_base::app` забезпечує дописування інформації у вихідний файл без перезапису вже існуючих даних. Це дозволяє зберігати результати кількох запусків програми у одному файлі, що може бути корисно для збору та аналізу великої кількості даних.

Також важливо відзначити, що для коректної роботи програми необхідний заголовковий файл `Modules.h` з описом функції `s_calculation`. Програма демонструє базові принципи взаємодії між файлами програми, оброблення числових даних та роботу з файловою системою, які є важливими аспектами програмування.

Програма також відображає важливість розуміння і використання різних типів даних та уміння ефективно застосовувати ці знання для вирішення

конкретних задач, таких як запис у файл вихідних даних у зручному для аналізу форматі.

Крім технічних аспектів, важливою є також здатність програми дописувати дані до існуючого файлу без його перезапису. Це може бути особливо корисним у ситуаціях, коли потрібно зберігати історичні дані або накопичувати результати з часом для подальшого аналізу.

Отриманні артефакти:

1) використання бібліотек: вод використовує стандартні бібліотеки `<iostream>`, `<fstream>`, а також заголовковий файл `"ModulesKlui.h"` для включення необхідних модулів.

2) оголошення змінних: оголошені змінні `b, x, y, z`, які будуть використані для зберігання введених користувачем значень.

3) введення користувачем значень: користувачеві пропонується ввести значення `x, y, z` та `b` з клавіатури за допомогою стандартного вводу `cin`.

4) робота з файлом: відкривається файл `"output.txt"` для дозапису результатів. Якщо відкриття файлу вдалося, результати обчислення функції `s_calculation(x, y, z)` записуються у файл, а також число `b` у двійковому коді за допомогою `bitset`.

Отриманні результати у задачі 10.3 включають:

1) результати обчислення функції `s_calculation`: у вихідний текстовий файл записуються результати виконання функції `s_calculation` з переданими аргументами `x, y, z`.

2) представлення числа `b` у двійковому коді: Після результатів обчислення функції `s_calculation` у файл також записується число `b` у двійковому форматі.

Ці результати відповідають вимогам задачі, де потрібно було записати обчислені значення та представлення числа у двійковому коді у вихідний файл.

На основі наведеного лістингу програми та виведеного результату, можна сформулювати наступні обґрунтовані висновки для задачі 10.3:

✓ *логічна структура програми:* програма складається з основної функції `main()`, де спочатку користувачеві пропонується ввести значення `x, y, z` та `b`. Після цього відкривається файл для дозапису результатів. У випадку успішного

відкриття файлу, обчислені значення функції s_calculation() та представлення числа b у двійковому вигляді записуються у файл. Нарешті, програма завершує свою роботу.

✓ *правильність розрахунків:* результати обчислення функції s_calculation() та представлення числа b у двійковому форматі відповідають вимогам задачі. Коректність обчислень перевірена та підтверджена.

✓ *тестування та стабільність:* програма відпрацьовує без помилок та недоліків при різних вхідних значеннях x, y, z та b. Вона стабільна та надійна у роботі, не викликає виникнення винятків або аварійних ситуацій.

✓ *читабельність та структурованість коду:* код має чітку структуру та правильно розділені функції. Імена змінних добре підібрані і відповідають їхньому призначенню. Коментарі над фрагментами коду зрозумілі та допомагають розібратися в його роботі.

Отже, на підставі аналізу коду та результатів виконання програми у задачі 10.3 можна зробити висновок, що програма ефективно вирішує поставлену задачу та відповідає всім вимогам.

Особисті враження від процесу виконання завдань лабораторної роботи були позитивними. Під час виконання завдань 1 і 2 - мала можливість детально ознайомитися з конкретними аспектами програмування, зокрема роботи з рядками, роботи з файлами та операціями над символами. Процес виконання задач 10.1, 10.2 та 10.3 був цікавим та пізнавальним, оскільки дозволив застосувати отримані знання на практиці.

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

1. Сутність понять області видимості об'єктів та простору імен у мові програмування C++ полягає у визначенні доступності та організації ідентифікаторів у програмі. Область видимості визначає, де ідентифікатор може

бути доступним у програмі, в той час як простір імен дозволяє структурувати ідентифікатори, уникнути конфліктів імен та полегшити роботу з великими програмами.

2. Час життя локального об'єкта (змінної) обмежений часом виконання області видимості, у межах якої він був оголошений. Приклад оголошення локальної та глобальної змінних одного зі складових типів мови C++:

```
#include <iostream>

// Глобальна змінна
int globalVariable = 10;

int main() {
    // Локальна змінна
    int localVariable = 5;

    std::cout << "Глобальна змінна: " << globalVariable << std::endl;
    std::cout << "Локальна змінна: " << localVariable << std::endl;

    return 0;
}
```

У цьому прикладі `globalVariable` - глобальна змінна, оголошена поза будь-якою функцією або блоком коду, тому вона має глобальну область видимості. `localVariable` - локальна змінна, оголошена всередині функції `main()`, тому її область видимості обмежена цією функцією.

3. У мові програмування C++, область видимості змінної, оголошеної у циклі, обмежена самим циклом. Це означає, що змінна буде доступна лише всередині тіла циклу, а поза ним - недоступна. Область видимості охоплює блок коду, у якому змінна була оголошена. Наведу приклад:

```
      #include <iostream>

int main() {
    for (int i = 0; i < 3; ++i) {
        std::cout << "Змінна i в циклі: " << i << std::endl;
    }

    // Наступний рядок спричинить помилку компіляції, оскільки змінна i недоступна
тут
    // std::cout << "Змінна i поза циклом: " << i << std::endl;

    return 0;
}
```

У цьому прикладі змінна `i` оголошена в циклі `for`. Тому змінна `i` буде доступна лише всередині циклу. Поза циклом, спроба звернутися до `i` призведе до

помилки компіляції, оскільки змінна `i` вже вийшла з області видимості. Таким чином, область видимості змінної, оголошеної у циклі, обмежена цим циклом.

4. Основним призначенням директива `extern` є повідомлення компілятору про існування цих змінних чи функцій без їх фактичного визначення. Це дозволяє компілятору правильно обробляти використання цих ідентифікаторів у поточному модулі. Звичайним використанням директиви `extern` є робота з міжмодульними змінними або функціями, наприклад:

```
// File1.cpp
#include <iostream>

// Оголошення змінної, яка знаходиться в іншому файлі
extern int globalVariable;

// Оголошення функції, яка знаходиться в іншому файлі
extern void externalFunction();

int main() {
    // Використання глобальної змінної та функції з іншого файлу
     std::cout << "Значення глобальної змінної з іншого файлу: " << globalVariable
<< std::endl;
    externalFunction();
    return 0;
}


// File2.cpp
#include <iostream>

// Визначення змінної, яка буде використана у іншому файлі
int globalVariable = 42;

// Визначення функції, яка буде використана у іншому файлі
void externalFunction() {
    std::cout << "Функція з іншого файлу" << std::endl;
}
```

У цьому прикладі в файлі `File1.cpp` ми використовуємо директиву `extern`, щоб оголосити глобальну змінну `globalVariable` та функцію `externalFunction`, які визначені у файлі `File2.cpp`. Під час компіляції та поєднання програми, компілятор знатиме про існування цих змінних та функцій, але їх фактичне визначення буде виконуватися у файлі `File2.cpp`.

5. Відмінність між масивом і рядком у мові програмування C/C++ з погляду реалізації їх оброблення полягає у наступному. *Масив*: масив представляє собою послідовність елементів одного типу, які розташовані в пам'яті поряд один з одним. Оброблення масивів включає в себе роботу з індексами, доступ до елементів за їхніми номерами та здійснення операцій з цілим масивом. Для

масивів немає стандартних функцій бібліотеки C/C++, але є можливість створювати власні функції для обробки масивів. *Рядок:* рядок є послідовністю символів, де кожен символ є типом `char`, і завершується нуль-символом `'\0'`. Оброблення рядків включає в себе використання функцій стандартної бібліотеки C/C++, таких як `strlen, strcpy, strcat,` тощо, які дозволяють виконувати операції з рядками. Рядок можна розглядати як масив символів `char`, але з додатковою умовою закінчення нуль-символом, що робить його рядком.

6. Об'єднання у мові програмування C/C++ дозволяє зберігати різні типи даних в одній і тій же пам'яті. Синтаксис оголошення та запису об'єднання виглядає так:

синтаксис оголошення об'єднання у C:

```
union UnionName {
  type1 member1;
  type2 member2;
  // Додаткові члени
};
```

синтаксис оголошення об'єднання у C++:

```
union UnionName {
    type1 member1;
    type2 member2;
    // Додаткові члени
};
```

У цих оголошеннях `UnionName` - це ім'я об'єднання, `type1, type2` - типи даних членів об'єднання, а `member1, member2` - назви членів.

7. Виняток (exception) у мові програмування - це об'єкт, що виникає в результаті помилки або несподіваної ситуації під час виконання програми. У мові програмування C++, оброблення винятків здійснюється за допомогою конструкції `"try-catch"`

8. Доступ до елементів масивів в мовах C/C++ здійснюється за допомогою операції індексації (`[]`) і вказівників. Для одновимірних масивів доступ до елементів здійснюється за одним індексом, а для двовимірних масивів - за двома індексами.

Оголошення одновимірного масиву: `type arrayName[size];`

Оголошення двовимірного масиву: `type arrayName[rows][columns];`

Де:

`type` - тип даних, з якого складаються елементи масиву.
`arrayName` - ім’я масиву.
`size` - кількість елементів у випадку одновимірного масиву.
`rows,columns` - кількість рядків та стовпців у випадку двовимірного масиву.

9. Функції з заголовного файлу `cstring` (або `string.h` у стандарті C) призначені для роботи з рядками символів в мові програмування C/C++. Ось короткий опис кожної з них:

`strstr`: повертає вказівник на перше входження підрядка в рядку.

`strlen`: повертає довжину рядка (кількість символів до завершального символа `'\0'`).

`strcpy`: копіює рядок з одного місця в інше.

`strncpy`: копіює задану кількість символів з одного рядка в інший.

`strcat`: додає один рядок в кінець іншого.

`strncat`: додає задану кількість символів одного рядка в кінець іншого.

`strcmp`: порівнює два рядки і повертає значення, яке вказує на їхню відмінність.

`strncmp`: порівнює задану кількість символів двох рядків.

`stricmp`: порівнює два рядки без врахування регістру символів.

`strnicmp`: порівнює задану кількість символів двох рядків без врахування регістру символів.

`strchr`: повертає вказівник на перше входження певного символа в рядок.

`strcspn`: повертає довжину префіксу рядка, що містить тільки символи, які не входять в другий рядок.

`strspn`: повертає довжину префіксу рядка, що містить тільки символи, які входять в другий рядок.

`strpbrk`: повертає вказівник на перший входження будь-якого символа з другого рядка в першому рядку.

`atof`: перетворює рядок у дійсне число (`double`).

`atoi`: перетворює рядок у ціле число (`int`).

`atol`: перетворює рядок у довге ціле число (`long`).

10. Структури у мовах програмування C та C++ призначені для об'єднання різних типів даних під одним ім'ям, що дозволяє створювати складніші типи даних. Вони дозволяють організувати дані за логічними групами та забезпечують зручну передачу та обробку об'єктів інформації в програмах. У відмінність від масивів, які містять лише однорідні елементи, структури можуть містити різні типи даних. Доступ до елементів масиву здійснюється за допомогою індексів, тоді як доступ до полів структури здійснюється за допомогою імен. Синтаксис:

```
// Оголошення структури
struct structName {
    type1 member1;
    type2 member2;
    // інші члени
};

// Оголошення екземпляру структури
structName objectName;

// Оголошення та ініціалізація екземпляру структури
struct structName {
    type1 member1;
    type2 member2;
    // інші члени
```

Де,

`structName` - це ім'я структури.

`type1, type2` - це типи даних членів структури.

`member1, member2` - це імена членів структури.

11. Член (поле) структури у мовах програмування C/C++ - це змінна, яка знаходиться в складі структури і має своє власне ім'я та тип даних. Він використовується для зберігання даних, що характеризують об'єкт, який представляє цю структуру. Доступ до членів структури здійснюється за допомогою оператора крапки (`.`), або через оператор стрілки (`->`) в C++, якщо ми маємо справу з вказівником на структуру.

12. Дескриптор структури (struct) - це вказівник на структуру. Він утворюється шляхом визначення змінної, яка є вказівником на об'єкт структури. Використовується програмістом для отримання доступу до членів структури, використовуючи цей вказівник.

13. Для оголошення змінної типу структура, якщо означений тип описано у заголовковому файлі, слід включити даний заголовний файл за допомогою директиви `#include` та оголосити змінну типу структура за допомогою ім'я цього означеного типу.

14. Приклад оголошення структури, один з членів якої є структурою у мові програмування C++:

```
#include <iostream>

// Оголошення структури
struct Address {
    std::string city;
    int postalCode;
};

// Структура, що містить адресу та ім'я
struct Person {
    std::string name;
    Address address; // Член структури типу Address
};

int main() {
    // Оголошення та ініціалізація об'єкту типу Person
    Person person1;
    person1.name = "John";
    person1.address.city = "New York";
    person1.address.postalCode = 10001;

    // Виведення інформації про особу
    std::cout << "Name: " << person1.name << std::endl;
    std::cout << "City: " << person1.address.city << std::endl;
    std::cout << "Postal Code: " << person1.address.postalCode << std::endl;

    return 0;
}
```

У цьому прикладі структура `Address` містить два члени: `city` та `postalCode`. Потім в структурі `Person` ми маємо ще один член з ім'ям `address`, який є структурою типу `Address`. Таким чином, `address` - це член структури `Person`, який є структурою.

15. У мовах програмування C/C++ можна оголосити змінну множинного типу за допомогою використання ключового слова `struct`. Ось синтаксис оголошення змінної множинного типу:

```
struct StructName {
    // Поля (члени) структури
    type1 member1;
    type2 member2;
    // інші члени
};

StructName variableName; // Оголошення змінної множинного типу
```

Операції, які можна виконувати над змінною множинного типу в мовах, включають: 1) можна отримувати доступ до окремих членів структури за допомогою оператора крапки (.) або оператора стрілки (->), якщо ми маємо справу з вказівником на структуру. 2) Значення можна присвоювати членам структури простим присвоюванням. 3) Можна передавати множинного типу у функцію як аргумент. 4) Функція може повертати множинного типу в якості результату. 5) Множинного типу можна ініціалізувати при її оголошенні або пізніше за допомогою фігурних дужок. 6) Множинного типу можна копіювати за допомогою оператора присвоювання або використовуючи конструктор копіювання

Відмінності між змінними типу масив і типу множина: 1) Масиви мають фіксований розмір, визначений під час компіляції, тоді як типу множина можуть містити різні типи даних та не мають фіксованого розміру. 2) Масиви зазвичай використовуються для зберігання групи однотипних елементів, тоді як типу множина дозволяють організовувати дані в логічні групи, які можуть містити різні типи даних. 3) Доступ до елементів масиву здійснюється за допомогою індексів, тоді як доступ до членів типу множина здійснюється за допомогою оператора крапки (.) або оператора стрілки (->), якщо ми маємо справу з вказівником на структуру.

16. У мовах програмування C/C++ існують два типи перетворень типів: явні та неявні. Неявні перетворення типів це автоматичні перетворення типів, які відбуваються без явного вказівника програміста. Вони відбуваються автоматично компілятором у певних ситуаціях, коли потрібно привести один тип даних до іншого для виконання певної операції. Явні перетворення типів це перетворення,

які вказуються програмістом явно за допомогою спеціальних операторів чи функцій. Вони використовуються, коли необхідно перетворити один тип даних у інший тип, і це не відбувається автоматично компілятором. В мові C++ також є ключове слово `static_cast`, яке використовується для явного перетворення типів.

17. За допомогою об'єкта `ofstream` здійснюється запис даних у файловий потік (файл) у мові програмування C++. Для цього використовується оператор вставки `<<`, який передає дані у файл через об'єкт `ofstream`. Приклад оголошення потокового об'єкта `ofstream` з використанням заголовного файла `fstream` у мові програмування C++:

```
#include <iostream>
#include <fstream>

int main() {
    // Оголошення об'єкта ofstream з використанням заголовного файла fstream
    std::ofstream outputFile;

    // Відкриття файлу для запису
    outputFile.open("output.txt");

    // Перевірка, чи відбулося відкриття файлу успішно
    if (!outputFile.is_open()) {
        std::cerr << "Failed to open the file!" << std::endl;
        return 1; // Повернення ненульового значення у разі помилки
    }

    // Запис даних у файл
    outputFile << "Hello, world!" << std::endl;

    // Закриття файлу
    outputFile.close();

    // Виведення повідомлення про успішне завершення запису
    std::cout << "Data has been written to the file successfully." << std::endl;

    return 0;
}
```

У цьому прикладі ми використали заголовний файл `<fstream>`, який містить визначення класу `ofstream` для роботи з файлами. Об'єкт `outputFile` оголошений як `ofstream` і використовується для запису даних у файл.

18. У мові програмування C++, файлові потоки, об'єкти яких оголошені типом `fstream, ofstream, ifstream` відповідно відкриваються у різних режимах за замовчуванням:

`fstream` - цей тип файлового потоку призначений для роботи з файлами у режимі зчитування та запису одночасно. Тобто, якщо відкрити файл у режимі `fstream`, ви зможете зчитувати дані з файлу та записувати нові дані у нього.

`ofstream` - цей тип файлового потоку призначений для запису даних у файл. Якщо відкрити файл у режимі `ofstream`, ви зможете записувати дані у файл, але не зможете зчитувати існуючі дані з нього.

`ifstream` - цей тип файлового потоку призначений для зчитування даних з файлу. Якщо відкрити файл у режимі `ifstream`, ви зможете зчитувати дані з файлу, але не зможете записувати в нього нові дані.

При відкритті файлу за допомогою об'єкта `fstream`, `ofstream`, `ifstream` можна вказати режим відкриття в явному вигляді, використовуючи аргумент `std::ios::openmode` у методі open()

19. Призначення функцій-членів потокового об’єкта `ifstream` у мові програмування C++:

- `open()`: ця функція-член використовується для відкриття файлу для зчитування даних. Вона приймає як аргумент рядок, що містить шлях до файлу, який потрібно відкрити, а також, за необхідності, додаткові параметри, такі як режим відкриття файлу. Після виклику цієї функції потоковий об'єкт буде готовий до зчитування даних з файлу.

- `eof()`: ця функція-член перевіряє, чи досягнуто кінця файлу. Вона повертає `true`, якщо вказівник потоку зчитування даних досягнув кінця файлу, і `false` в іншому випадку. Використовується, наприклад, для перевірки умови завершення зчитування даних з файлу в циклі.

- `close()`: ця функція-член використовується для закриття файлу, що відкритий для зчитування. Після виклику цієї функції доступ до файлу для зчитування буде припинено, і ресурси, пов'язані з цим файлом, будуть звільнені. Це важливо зробити після завершення роботи з файлом, щоб уникнути витоку ресурсів.

20. Пояснення кожної з перелічених констант режимів відкриття файлових потоків у мові програмування C++:

`ios_base::app`: ця константа вказує, що під час відкриття файлу дані будуть дописуватися в кінець файлу. Якщо файл вже існує, вказівник зміщується в кінець файлу перед початком запису. Якщо файл не існує, він буде створений.

`ios_base::ate`: ця константа вказує, що під час відкриття файлу позиція вказівника встановлюється в кінець файлу. Тобто, читання та запис відбуватимуться в кінці файлу. Якщо файл не існує, він буде створений.

`ios_base::ate`: ця константа вказує, що під час відкриття файлу позиція вказівника встановлюється в кінець файлу. Тобто, читання та запис відбуватимуться в кінці файлу. Якщо файл не існує, він буде створений.

`ios_base::in`: ця константа вказує, що файл буде відкритий тільки для зчитування даних. Це означає, що ви не зможете записати нові дані в файл, але зможете зчитувати вже існуючі дані з нього.

`ios_base::out`: ця константа вказує, що файл буде відкритий тільки для запису даних. Ви можете записувати нові дані у файл, але не зможете зчитувати вже існуючі дані з нього.

`ios_base::trunc`: ця константа вказує, що під час відкриття файлу вміст файлу буде видалений, якщо файл вже існує. Це означає, що файл буде обрізаний (очищений), і буде створений новий порожній файл. Якщо файл не існує, він буде створений.

Ці константи використовуються разом з методом `open()` для вказання режиму відкриття файлу

ВІДПОВІДЬ НА КОНТРОЛЬНІ ЗАПИТАННЯ І ЗАВДАННЯ

1. Блок-контроль `try-throw-catch` в мові програмування C++ використовується для обробки винятків, тобто ситуацій, коли виникає помилка або непередбачувана ситуація під час виконання програми. Основне призначення цього блоку - забезпечення можливості перехоплення та обробки винятків, що

виникають під час виконання програми, з метою збереження нормальної роботи програми або попередження її аварійного завершення. Синтаксис:

```
try {
    // Код, який може викинути виняток
    throw SomeException(); // Виклик конструктора винятка
} catch (const SomeException& e) {
    // Обробник винятку
    std::cerr << "Caught an exception: " << e.what() << std::endl;
} catch (...) {
    // Обробник винятку для всіх інших типів винятків
    std::cerr << "Caught an unknown exception." << std::endl;
}
```

Основні елементи блоку `try-throw-catch`:

- `try`: у цьому блоку розміщується код, який може викинути виняток.

- `throw`: ключове слово `throw` використовується для генерації (викидання) винятку. Ви можете викинути будь-який тип об'єкта, включаючи власні класи винятків.

-`catch`: у цьому блоку розміщуються обробники винятків. Вони відповідають за перехоплення та обробку винятків, які виникають у блоку `try`. Кожен `catch` визначає обробник для певного типу винятка. Можливо мати кілька блоків `catch` для обробки різних типів винятків.

-`... (ellipsis)`: це спеціальний блок `catch`, який перехоплює будь-який інший тип винятку, який не було перехоплено раніше. Він може бути корисним для обробки винятків, які неочікувано з’являються.

2. Міжмодульна змінна у мові програмування C++ - це змінна, яка використовується у кількох модулях (файлах) програми. Для цього змінну потрібно оголосити в одному модулі (зазвичай у заголовковому файлі) та зробити її доступною для інших модулів. Одним із способів роботи з міжмодульними змінними є використання ключового слова `extern`. Ось приклад:

Файл1.cpp:

```
// Файл1.cpp
#include <iostream>

// Оголошення міжмодульної змінної
extern int globalVariable;

int main() {
```

```
    // Використання міжмодульної змінної
       std::cout << "Значення міжмодульної змінної: " << globalVariable << std::endl;
    return 0;
}
```

Файл2.cpp:

```
// Файл2.cpp
#include <iostream>

// Оголошення та визначення міжмодульної змінної
int globalVariable = 42;
```

У файлі Файл1.cpp використовуємо ключове слово `extern` для оголошення міжмодульної змінної `globalVariable`. Це оголошення дозволяє компілятору знати про існування цієї змінної, але не вимагає фактичного визначення її значення у цьому файлі.

У файлі Файл2.cpp визначаємо та ініціалізуємо міжмодульну змінну `globalVariable`.

Під час компіляції обидва файли будуть оброблені, і значення `globalVariable` з файлу Файл2.cpp буде доступне для використання у файлі Файл1.cpp. При виконанні програми, вивід `std::cout` у файлі Файл1.cpp відобразить значення міжмодульної змінної.

3. Об'єкти (змінні, типи, константи тощо), описані в тілі функції `main` в C++, матимуть локальну область видимості. Це означає, що вони будуть доступні лише всередині тіла функції main та будуть існувати лише під час виконання цієї функції.

4. Тип `enum` та масиви є двома різними конструкціями у мові програмування C++, які використовуються для зберігання та роботи з даними. Порівняємо їх за декількома ключовими аспектами:

1) Тип даних:

- `enum` - це перерахувальний тип даних, який дозволяє задати множину іменованих цілочисельних констант. Кожне ім'я в `enum` має відповідну цілочисельну константу.

- масив - це набір послідовних елементів одного типу даних. Елементами масиву можуть бути будь-які типи даних, включаючи цілі числа, рядки, плаваючі точкові числа тощо.

2) Оголошення:

- оголошення `enum` визначається за допомогою ключового слова `enum`. Наприклад: `enum Color { RED, GREEN, BLUE };`

- оголошення масиву визначається за допомогою типу елементів масиву та ім'я масиву. Наприклад: `int numbers[5]`

3) Робота з елементами:

- елементи `enum` можуть використовуватися для задання конкретних станів або значень, які потім можна використовувати в програмі для управління поведінкою.

- елементи масиву можуть використовуватися для зберігання послідовної колекції даних одного типу. Вони можуть бути звернуті за допомогою індексів.

4) Розмір інформації:

- по суті, `enum` використовується для зручної роботи з іменованими значеннями, інформація про які зазвичай зберігається як цілі числа. Таким чином, кожен елемент `enum` займає певну кількість байтів в пам’яті.

- розмір масиву визначається кількістю його елементів та розміром кожного елемента. Розмір масиву може бути статичним або динамічно змінюваним.

5) Доступ до елементів:

- доступ до елементів `enum` здійснюється через ім'я константи.

- доступ до елементів масиву здійснюється за допомогою індексів.

6) Значення за замовчуванням:

- першому елементу перерахування типу `enum` автоматично надається значення 0, а кожний наступний елемент отримує значення, яке на 1 більше за попередній.

- значення за замовчуванням у масиві визначаються його типом даних, наприклад, для масиву цілих чисел значення за замовчуванням будуть нулі.

Обидва ці конструкції мають свої властивості та застосування в залежності від контексту. `enum` зазвичай використовується для створення переліків констант, тоді як масиви використовуються для зберігання послідовності значень того ж типу.