Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

Механіко-технологічний факультет

ЗВІТ

ПРО ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ № 11

з навчальної дисципліни

“Базові методології та технології програмування”

КОМАНДНА РЕАЛІЗАЦІЯ ПРОГРАМНИХ ЗАСОБІВ ОБРОБЛЕННЯ

ДИНАМІЧНИХ СТРУКТУР ДАНИХ ТА БІНАРНИХ ФАЙЛІВ

ЗАВДАННЯ ВИДАВ

доцент кафедри кібербезпеки та програмного забезпечення

Доренський О. П.

[https://github.com/odorenskyi/](https://github.com/odorenskyi/Dmytro-Parkhomenko-KB18)

ВИКОНАВ

студент академічної групи КІ - 24

Савельєв В. В.

ПЕРЕВІРИВ

ст. викладач кафедри кібербезпеки   
та програмного забезпечення

Коваленко А.С

Кропивницький – 2025

**ТЕМА:** Командна реалізація програмних засобів оброблення динамічних структур даних та бінарних файлів

**МЕТА** Полягає у набутті ґрунтовних вмінь і практичних навичок командної (колективної) реалізації програмного забезпечення, розроблення функцій оброблення динамічних структур даних, використання стандартних засобів С++ для керування динамічною пам’яттю та бінарними файловими потоками.

**ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ**

Створити базу даних “Відділ кадрів: особова картка працівника, ф. П-2"(дата заповнення, табельний номер, ідентифікаційний номер, стать, вид роботи /основна, за сумісництвом/ ПІБ працівника, дата народження, громадянство, освіта, останнє місце роботи і посада, стаж роботи /днів, місяців, років/, місце проживання, паспортні дані, додаткові відомості, дата і причина звільнення).

**СКЛАД КОМАНДИ:**

Андрій Федотов - <https://github.com/odorenskyi/Fedotov-Andrii-KI24>

Володимир Савлеьєв - <https://github.com/odorenskyi/Saveliev-Volodymyr-KI24>

Балан Микола - <https://github.com/odorenskyi/Balan-Mykola-KI24>

Таблиця 1 – данні команди

| Учасник | Функції | Опис |
| --- | --- | --- |
| Андрій Федотов | saveToFile() | Збереження у файл. |
|  | searchByCode() | Пошук за кодом |
| Володимир Савлеьєв | addBenefit() | Додавання нового запису |
|  | deleteByCode() | Видалення запису |
| Балан Микола | displayAll() | Виведення всіх записів. |
|  | exportToTextFile() | Експорт у текстовий файл |

**ВИМОГИ ДО СИСТЕМИ**

**ФУНКЦІОНАЛЬНІ ВИМОГИ**

1. Програма повинна створювати електронний довідник пільг, що включає поля: Код пільги (ціле число), Зміст пільги (рядок до 100 символів), Документ (рядок до 100 символів), № документа (рядок до 50 символів), Дата прийняття документа (рядок у форматі РРРР-ММ-ДД), Початок дії пільги (рядок у форматі РРРР-ММ-ДД), Кінець дії пільги (рядок у форматі РРРР-ММ-ДД), Ознака цільового призначення (рядок до 50 символів), Коментар (рядок до 100 символів).
2. Програма має забезпечувати пошук запису в довіднику за введеним кодом пільги з подальшим виведенням усіх полів знайденого запису.
3. Програма має надавати можливість виведення всіх записів довідника на екран.
4. Програма має підтримувати експорт усіх записів довідника у текстовий файл із заданою назвою.
5. Програма має дозволяти додавання нового запису в довідник із валідацією введених даних (наприклад, формату дати).
6. Програма має забезпечувати видалення запису з довідника за введеним кодом пільги.
7. Програма повинна автоматично зберігати всі дані довідника у бінарний файл при завершенні роботи (опція "Вихід та збереження").
8. Програма має автоматично завантажувати дані довідника з бінарного файлу під час запуску.

**АРХІТЕКТУРА СИСТЕМИ**

1. Система реалізована як консольна програма на мові C++ з використанням стандартної бібліотеки (STL).
2. Програма складається з кількох модулів, які розділені на окремі файли для підвищення модульності та зручності модифікації.
3. Дані зберігаються у бінарному файлі (benefit\_data.bin) для ефективного доступу та збереження.
4. Взаємодія з користувачем здійснюється через текстове меню з підтримкою кириличного введення та виведення (UTF-8 кодування).

**ОСНОВНІ КОМПОНЕНТИ**

1. **Структура даних (**struct\_type\_project\_1.h**)**: Описує структуру Benefit із полями для зберігання інформації про пільги, включаючи конструктор для ініціалізації.
2. **Модуль Balan (**ModulesBalan.h**)**: Містить функції для виведення всіх записів (displayAll) та експорту в текстовий файл (exportToTextFile).
3. **Модуль Fedotov (**ModulesFedotov.h**)**: Містить функції для збереження даних у файл (saveToFile) та пошуку за кодом (searchByCode).
4. **Модуль Savelyev (**ModulesSavelyev.h**)**: Містить функції для додавання нового запису (addBenefit) та видалення запису (deleteByCode).
5. **Основна програма (**main.cpp**)**: Реалізує головне меню, функцію завантаження даних (loadFromFile) та керування роботою програми, включаючи налаштування кодування (UTF-8).

**СХЕМА ВЗАЄМОДІЇ МОДУЛІВ**  
main.cpp --> ModulesFedotov.h (loadFromFile, saveToFile);

main.cpp --> ModulesBalan.h (displayAll, exportToTextFile);

main.cpp --> ModulesSavelyev.h (addBenefit, deleteByCode, searchByCode);

ModulesFedotov.h --> struct\_type\_project\_1.h;

ModulesBalan.h --> struct\_type\_project\_1.h;

ModulesSavelyev.h --> struct\_type\_project\_1.h;

**ПЛАН РОБІТ**

1. **Аналіз вимог (3.1)**  
   Опис: Визначення вимог до довідника пільг (пошук, виведення, додавання, видалення, завантаження/збереження даних) та аналіз формату даних.
2. **Проектування:**

* **Специфікації ПЗ (3.2)**  
  Опис: Розробка специфікації функціональності, меню та формату бінарного файлу.
* **Вибір структури даних (3.3)**  
  Опис: Визначення структури Benefit і використання вектора для зберігання.
* **Створення struct\_type\_project\_1.h (3.4)**  
  Опис: Реалізація файлу з структурою Benefit і конструктором.

1. **Реалізація:**

* **Розподіл завдань (3.5)**  
  Опис: Розподіл між ModulesBalan.h (виведення/експорт), ModulesFedotov.h (збереження/пошук), ModulesSavelyev.h (додавання/видалення).
* **Кодування модулів**  
  Опис: Написання коду для функцій (displayAll, exportToTextFile, saveToFile, searchByCode, addBenefit, deleteByCode) та інтеграція з main.cpp.

1. **Тестування:**

* **Перевірка завантаження/збереження**  
  Опис: Тестування loadFromFile і saveToFile з файлом benefit\_data.bin.
* **Тестування операцій із записами**  
  Опис: Перевірка додавання, видалення, пошуку, виведення та експорту з валідацією даних.

1. **Затвердження плану викладачем (3.7)**

**ПИКЛАД БАЗИ ДАННИХ ЕЛЕКТРОННИЙ ДОВІДНИК ПІЛЬГ**

Таблиця 2 – приклад бази данних

| **Код пільги** | **Зміст пільги** | **Документ** | **№ документа** | **Дата прийняття документа** | **Початок дії пільги** | **Кінець дії пільги** | **Ознака цільового призначення** | **Коментар** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 11020025 | Сума коштів, що не підлягає оподаткуванню, сплаченими донорами крові України | Закон України "Про донорство крові та її компонентів" | 3349-ХІV | 28.12.1994 | 02.18.1997 | 31.12.2030 | 01.01.2025 | 15.05.2025 |
| 11020085 | Сума податку на прибуток, отриманих іноземних інвестицій, що не підлягає оподаткуванню | Податковий кодекс України | 2755-VI | 02.12.2010 | 01.04.2015 | 31.12.2028 | 01.06.2024 | 10.03.2025 |
| 11020086 | Річна сума податку на прибуток, отриманих іноземних інвестицій, що не підлягає оподаткуванню | Податковий кодекс України | 2755-VI | 02.12.2010 | 01.04.2015 | 31.12.2029 | 15.07.2023 | 20.04.2025 |
| 11020087 | Не підлягає оподаткуванню сума земельного податку за землі, надані в оренду | Закон України "Про внесення змін до Податкового кодексу України" | 197-VIII | 21.12.2016 | 01.04.2015 | 31.12.2026 | 01.03.2025 | 25.05.2025 |

**СТРУКТУРА** struct\_type\_project\_1.h  
#ifndef STRUCT\_TYPE\_PROJECT\_1\_H

#define STRUCT\_TYPE\_PROJECT\_1\_H

#include <cstring>

struct Benefit {

int benefitCode;

char description[100];

char documentName[100];

char documentNumber[50];

char documentDate[20];

char startDate[20];

char endDate[20];

char targetPurpose[50];

char comment[100];

Benefit() : benefitCode(0) {

std::memset(description, 0, sizeof(description));

std::memset(documentName, 0, sizeof(documentName));

std::memset(documentNumber, 0, sizeof(documentNumber));

std::memset(documentDate, 0, sizeof(documentDate));

std::memset(startDate, 0, sizeof(startDate));

std::memset(endDate, 0, sizeof(endDate));

std::memset(targetPurpose, 0, sizeof(targetPurpose));

std::memset(comment, 0, sizeof(comment));

}

};

#endif

**МОДУЛЬ** ModulesSavelyev

**ПОСТАНОВА ЗАДАЧІ**

Необхідно розробити модуль для електронного довідника пільг, який забезпечує додавання нового запису до вектора з валідацією введених даних, зокрема формату дат, і видалення запису за введеним кодом пільги з повідомленням про успішність операції або її відсутність.

**АНАЛІЗ ЗАДАЧІ**

Для реалізації модуля потрібно створити функцію додавання запису, яка запитує у користувача всі необхідні поля структури Benefit, перевіряє коректність введеного коду пільги та формату дат за допомогою регулярного виразу, додає запис до вектора і підтверджує операцію, а також функцію видалення, яка знаходить запис за кодом у векторі, видаляє його, якщо він існує, і виводить відповідне повідомлення, забезпечуючи при цьому захист від некоректних даних і підтримку кирилиці**.**

**РЕАЛІЗАЦІЯ БІБЛІОТЕКИ**

Вихідний файл ModulesSavelyev.cpp

#include "struct\_type\_project\_1.h"

#include <fstream>

#include <iostream>

#include <vector>

#include <cstring>

#include <locale>

#include <regex>

const std::string DATA\_FILE = "benefit\_data.bin";

// Додавання нового запису

void addBenefit(std::vector<Benefit>& list) {

Benefit b;

std::cout << "Введіть код пільги: ";

while (!(std::cin >> b.benefitCode) || b.benefitCode <= 0) {

std::cout << "Невірне значення. Введіть додатне ціле число: ";

std::cin.clear();

std::cin.ignore(1000, '\n');

}

std::cin.ignore();

std::cout << "Введіть зміст пільги: ";

std::cin.getline(b.description, sizeof(b.description));

std::cout << "Введіть назву документа: ";

std::cin.getline(b.documentName, sizeof(b.documentName));

std::cout << "Введіть номер документа: ";

std::cin.getline(b.documentNumber, sizeof(b.documentNumber));

std::regex datePattern(R"((\d{4})-(\d{2})-(\d{2}))");

std::cmatch match;

do {

std::cout << "Введіть дату прийняття документа (РРРР-ММ-ДД): ";

std::cin.getline(b.documentDate, sizeof(b.documentDate));

if (std::regex\_match(b.documentDate, match, datePattern)) {

int y = std::stoi(match[1]);

int m = std::stoi(match[2]);

int d = std::stoi(match[3]);

int maxDay[] = { 31,28,31,30,31,30,31,31,30,31,30,31 };

if ((y % 4 == 0 && y % 100 != 0) || (y % 400 == 0)) maxDay[1] = 29;

if (m >= 1 && m <= 12 && d >= 1 && d <= maxDay[m - 1]) break;

}

std::cout << "Некоректна дата. Повторіть.\n";

} while (true);

do {

std::cout << "Введіть дату початку дії пільги (РРРР-ММ-ДД): ";

std::cin.getline(b.startDate, sizeof(b.startDate));

if (std::regex\_match(b.startDate, match, datePattern)) {

int y = std::stoi(match[1]);

int m = std::stoi(match[2]);

int d = std::stoi(match[3]);

int maxDay[] = { 31,28,31,30,31,30,31,31,30,31,30,31 };

if ((y % 4 == 0 && y % 100 != 0) || (y % 400 == 0)) maxDay[1] = 29;

if (m >= 1 && m <= 12 && d >= 1 && d <= maxDay[m - 1]) break;

}

std::cout << "Некоректна дата. Повторіть.\n";

} while (true);

do {

std::cout << "Введіть дату закінчення дії пільги (РРРР-ММ-ДД): ";

std::cin.getline(b.endDate, sizeof(b.endDate));

if (std::regex\_match(b.endDate, match, datePattern)) {

int y = std::stoi(match[1]);

int m = std::stoi(match[2]);

int d = std::stoi(match[3]);

int maxDay[] = { 31,28,31,30,31,30,31,31,30,31,30,31 };

if ((y % 4 == 0 && y % 100 != 0) || (y % 400 == 0)) maxDay[1] = 29;

if (m >= 1 && m <= 12 && d >= 1 && d <= maxDay[m - 1]) break;

}

std::cout << "Некоректна дата. Повторіть.\n";

} while (true);

std::cout << "Введіть ознаку цільового призначення: ";

std::cin.getline(b.targetPurpose, sizeof(b.targetPurpose));

std::cout << "Введіть коментар: ";

std::cin.getline(b.comment, sizeof(b.comment));

list.push\_back(b);

std::cout << "Пільгу додано.\n";

}

// Видалення запису

void deleteByCode(std::vector<Benefit>& list, int code) {

if (code <= 0) {

std::cout << "Некоректний код пільги.\n";

return;

}

auto it = list.begin();

while (it != list.end()) {

if (it->benefitCode == code) {

it = list.erase(it);

std::cout << "Запис із кодом " << code << " вилучено.\n";

return;

}

++it;

}

std::cout << "Запис із кодом " << code << " не знайдено.\n";

}

Заголовковий файл ModulesSavelyev.h

#ifndef MODULESSAVELEV\_H

#define MODULESSAVELEV\_H

#include <vector>

#include "struct\_type\_project\_1.h"

// Додавання нового запису

void addBenefit(std::vector<Benefit>& list);

// Видалення запису

void deleteByCode(std::vector<Benefit>& list, int code);

#endif

**МОДУЛЬ** ModulesFedotov

**РЕАЛІЗАЦІЯ БІБЛІОТЕКИ**

Вихідний файл ModulesFedotov.cpp

#include "struct\_type\_project\_1.h"

#include <fstream>

#include <iostream>

#include <vector>

#include <cstring>

#include <locale>

const std::string DATA\_FILE = "benefit\_data.bin";

// Збереження у файл

void saveToFile(const std::vector<Benefit>& list) {

std::ofstream fout(DATA\_FILE, std::ios::binary);

if (!fout) {

std::cerr << "Помилка при відкритті файлу для запису.\n";

return;

}

for (const Benefit& b : list) {

if (b.benefitCode <= 0) continue; // Пропускаємо некоректні записи

fout.write(reinterpret\_cast<const char\*>(&b), sizeof(Benefit));

}

if (!fout.good()) {

std::cerr << "Сталася помилка при записі у файл.\n";

}

fout.close();

std::cout << "Дані збережено у файл: " << DATA\_FILE << "\n";

}

// Пошук за кодом

void searchByCode(const std::vector<Benefit>& list, int code) {

bool found = false;

for (const Benefit& b : list) {

if (b.benefitCode == code && b.benefitCode > 0) {

std::cout << "\nЗапис знайдено:\n";

std::cout << "Код пільги: " << b.benefitCode << "\n";

std::cout << "Зміст пільги: " << (b.description[0] ? b.description : "Немає") << "\n";

std::cout << "Документ: " << (b.documentName[0] ? b.documentName : "Немає") << "\n";

std::cout << "Номер документа: " << (b.documentNumber[0] ? b.documentNumber : "Немає") << "\n";

std::cout << "Дата прийняття документа: " << (b.documentDate[0] ? b.documentDate : "Немає") << "\n";

std::cout << "Початок дії пільги: " << (b.startDate[0] ? b.startDate : "Немає") << "\n";

std::cout << "Кінець дії пільги: " << (b.endDate[0] ? b.endDate : "Немає") << "\n";

std::cout << "Ознака цільового призначення: " << (b.targetPurpose[0] ? b.targetPurpose : "Немає") << "\n";

std::cout << "Коментар: " << (b.comment[0] ? b.comment : "Немає") << "\n";

found = true;

break;

}

}

if (!found) {

std::cout << "Запис із кодом " << code << " не знайдено.\n";

}

}

Заголовковий файл ModulesFedotov.h  
#ifndef MODULESFEDOTOV\_H

#define MODULESFEDOTOV\_H

#include <vector>

#include <string>

#include "struct\_type\_project\_1.h"

// Збереження у файл

void saveToFile(const std::vector<Benefit>& list);

// Пошук за кодом

void searchByCode(const std::vector<Benefit>& list, int code);

#endif

**МОДУЛЬ** ModulesBalan

**РЕАЛІЗАЦІЯ БІБЛІОТЕКИ**

Вихідний файл ModulesBalan.cpp

#include "struct\_type\_project\_1.h"

#include <fstream>

#include <iostream>

#include <vector>

#include <cstring>

#include <locale>

const std::string DATA\_FILE = "benefit\_data.bin";

// Виведення всіх записів

void displayAll(const std::vector<Benefit>& list) {

if (list.empty()) {

std::cout << "Довідник порожній.\n";

return;

}

std::cout << "\nУсі записи довідника:\n";

for (const Benefit& b : list) {

if (b.benefitCode <= 0) {

std::cout << "Пошкоджений запис (некоректний код пільги).\n";

continue;

}

std::cout << "-------------------------\n";

std::cout << "Код пільги: " << b.benefitCode << "\n";

std::cout << "Зміст пільги: " << (b.description[0] ? b.description : "Немає") << "\n";

std::cout << "Документ: " << (b.documentName[0] ? b.documentName : "Немає") << "\n";

std::cout << "Номер документа: " << (b.documentNumber[0] ? b.documentNumber : "Немає") << "\n";

std::cout << "Дата прийняття документа: " << (b.documentDate[0] ? b.documentDate : "Немає") << "\n";

std::cout << "Початок дії пільги: " << (b.startDate[0] ? b.startDate : "Немає") << "\n";

std::cout << "Кінець дії пільги: " << (b.endDate[0] ? b.endDate : "Немає") << "\n";

std::cout << "Ознака цільового призначення: " << (b.targetPurpose[0] ? b.targetPurpose : "Немає") << "\n";

std::cout << "Коментар: " << (b.comment[0] ? b.comment : "Немає") << "\n";

}

}

// Експорт у текстовий файл

void exportToTextFile(const std::vector<Benefit>& list, const std::string& filename) {

std::ofstream fout(filename);

if (!fout) {

std::cerr << "Не вдалося відкрити файл для запису: " << filename << "\n";

return;

}

for (const Benefit& b : list) {

if (b.benefitCode <= 0) continue; // Пропускаємо некоректні записи

fout << "Код пільги: " << b.benefitCode << "\n";

fout << "Зміст пільги: " << (b.description[0] ? b.description : "Немає") << "\n";

fout << "Документ: " << (b.documentName[0] ? b.documentName : "Немає") << "\n";

fout << "Номер документа: " << (b.documentNumber[0] ? b.documentNumber : "Немає") << "\n";

fout << "Дата прийняття документа: " << (b.documentDate[0] ? b.documentDate : "Немає") << "\n";

fout << "Початок дії пільги: " << (b.startDate[0] ? b.startDate : "Немає") << "\n";

fout << "Кінець дії пільги: " << (b.endDate[0] ? b.endDate : "Немає") << "\n";

fout << "Ознака цільового призначення: " << (b.targetPurpose[0] ? b.targetPurpose : "Немає") << "\n";

fout << "Коментар: " << (b.comment[0] ? b.comment : "Немає") << "\n";

fout << "---------------------------\n";

}

fout.close();

std::cout << "Експортовано у файл: " << filename << "\n";

}

Заголовковий файл ModulesBalan.h

#ifndef MODULESBALAN\_H

#define MODULESBALAN\_H

#include <vector>

#include <string>

#include "struct\_type\_project\_1.h"

// Виведення всіх записів

void displayAll(const std::vector<Benefit>& list);

// Експорт у текстовий файл

void exportToTextFile(const std::vector<Benefit>& list, const std::string& filename);

#endif

**ВИСНОВОКИ**

1. Модульність коду підвищує зручність його підтримки та розвитку.
2. Структури дозволяють логічно групувати різнорідні дані.
3. Об’єднання (union) економлять пам’ять, зберігаючи різні типи в одному місці.
4. Множини (set) дають змогу працювати з унікальними елементами.
5. Перелічення (enum) забезпечують зручне позначення сталих значень.
6. Масиви зручні для зберігання однотипних елементів.
7. Перетворення типів необхідні для коректної взаємодії різних даних.
8. Файлові потоки (fstream) використовуються для збереження та читання даних.
9. Читання з файлу потребує перевірки коректності відкриття файлу.
10. Запис у файл повинен бути організований для уникнення втрати даних.
11. Бінарний режим файлів зберігає точність даних без форматування.
12. Текстовий режим зручний для читання даних людиною.
13. Ізоляція логіки обробки даних у функціях підвищує читабельність коду.
14. Використання стандартних бібліотек спрощує роботу з рядками.
15. Рядкові операції важливі для обробки символьної інформації.
16. Обробка помилок при роботі з файлами є обов’язковою.
17. Ініціалізація структур забезпечує передбачувану поведінку.
18. Використання констант підвищує безпеку даних.
19. Іменовані константи роблять код зрозумілішим.
20. Використання циклів дозволяє ефективно опрацьовувати масиви.
21. Рекурсія в модулях не завжди є оптимальним рішенням.
22. Використання вказівників розширює можливості керування пам’яттю.
23. Динамічне виділення пам’яті потрібне для змінних розмірів даних.
24. Необхідно вчасно звільняти динамічну пам’ять для уникнення витоків.
25. Функції обробки даних мають чітко визначені інтерфейси.
26. Передача аргументів за посиланням економить ресурси.
27. Імпорт і експорт даних у файл підвищує гнучкість програм.
28. Перевірка вхідних даних підвищує надійність програм.
29. Логічна структура програмного модуля сприяє повторному використанню.
30. Конвенції іменування підвищують зрозумілість коду.
31. Коментарі допомагають підтримувати код у довгостроковій перспективі.
32. Відокремлення логіки від представлення покращує архітектуру.
33. Стандартні алгоритми STL можуть спростити обробку множин.
34. Функції форматування полегшують вивід структурованих даних.
35. Використання typedef або using робить код гнучкішим.
36. Використання enum класів покращує типобезпечність.
37. Валідація введених даних у файлах знижує ризик помилок.
38. Модульні тести допомагають перевірити коректність функцій.
39. Оптимізація читання/запису файлів підвищує продуктивність.
40. Буферизація вводу/виводу прискорює роботу з файлами.
41. Програма має коректно реагувати на відсутність файлу.
42. Форматування текстових файлів повинно бути узгодженим.
43. Використання структурованих типів підвищує надійність коду.
44. Множини ефективні для операцій об’єднання, перетину, різниці.
45. Об’єднання корисні, коли дані можуть бути різного типу, але не одночасно.
46. Використання constexpr покращує продуктивність.
47. Уважність при роботі з покажчиками запобігає аваріям.
48. Винятки (exceptions) можуть бути корисні для обробки помилок.
49. Стандартні потоки (cin, cout) можна використовувати разом з файловими.
50. Форматування виводу допомагає візуалізувати дані.
51. Розділення коду на заголовочні файли (.h) і реалізації (.cpp) підтримує структуру.
52. Використання namespace зменшує конфлікти імен.
53. Підхід «один файл — одна відповідальність» підвищує масштабованість.
54. Опис типів у структурах повинен бути зрозумілим і точним.
55. Масиви і вектори мають різні сфери застосування.
56. Використання std::vector краще за сирі масиви для динамічних даних.
57. Обробка символьних даних вимагає уваги до кодування.
58. Конвертація типів повинна бути явною для уникнення помилок.
59. Функції-члени структур можуть покращити організацію коду.
60. Оголошення та ініціалізація констант мають бути в одному місці.
61. Використання inline функцій підвищує швидкодію.
62. Застосування шаблонів спрощує роботу з різними типами.
63. Чіткий поділ відповідальностей між модулями підвищує якість.
64. Використання стандартних потоків робить код більш переносимим.
65. Програми мають бути готові до некоректних або пошкоджених файлів.
66. Автоматичне тестування допомагає виявляти помилки на ранніх стадіях.
67. Використання сучасних стандартів C++ (11/14/17/20) покращує код.
68. Рефакторинг допомагає підтримувати чистоту та ефективність коду.
69. Документування функцій підвищує їх зрозумілість.
70. Обробка помилок у файлових операціях є критичною.
71. Використання логування допомагає у відлагодженні.
72. Формат даних у файлах має бути узгодженим для сумісності.
73. Співпраця між модулями потребує чітких контрактів.
74. Застосування патернів проектування покращує архітектуру.
75. Практика та експерименти сприяють кращому засвоєнню матеріалу.

**ВІДПОВІДЬНА КОНТРОЛЬНІ ЗАПИТАННЯ**

1. **Призначення та синтаксис блоку try-throw-catch у C++**Блок try використовується для обгортання коду, у якому може виникнути виняткова ситуація (помилка). За допомогою оператора throw створюється виняток, який передається до відповідного блоку catch для обробки. Це механізм обробки помилок, що дозволяє відокремити логіку обробки помилок від основного коду.
2. **Приклад опису й використання міжмодульної змінної**Міжмодульна змінна — це змінна, оголошена в одному модулі з ключовим словом extern для доступу з інших модулів. В одному файлі вона визначається (ініціалізується), в інших — лише оголошується через extern, що дозволяє уникнути дублювання.
3. **Область видимості об’єктів, описаних у функції main**Змінні, типи, константи, оголошені всередині функції main, мають локальну область видимості — вони доступні лише в межах тіла цієї функції.
4. **Порівняльний аналіз enum та масиву**enum — це перелік іменованих сталих значень, які мають ціле числове представлення, призначений для зручного позначення набору сталих. Масив — це набір однотипних елементів, доступних за індексом. Enum не зберігає множину значень, а лише символічні константи, тоді як масив — це структура для зберігання множини даних.
5. **Визначення потоку та файлового потоку**Потік — це послідовність даних, що передаються для вводу або виводу. Файловий потік — це спеціальний потік, що забезпечує доступ до даних, записаних у файл, і відрізняється від стандартних потоків (наприклад, клавіатура, консоль) тим, що працює з довготривалим збереженням інформації.
6. **Універсальний алгоритм читання/запису даних з/у файл**Відкрити файл у відповідному режимі (читання/запис), перевірити успішність відкриття, виконати операції читання або запису, закрити файл, обробити можливі помилки.
7. **Об’єм текстового файлу з числом π з точністю 106 знаків після коми**Число π з цією точністю буде містити 1 символ перед комою, 1 крапку, 106 цифр після коми — разом 108 символів. Кожен символ у текстовому файлі зазвичай займає 1 байт, отже об’єм файлу приблизно 108 байтів.
8. **Реалізація перевірки статусу відкриття файлу та мета**Перевіряють, чи файл відкрився успішно (наприклад, за допомогою методів або операторів потоків). Мета — уникнути помилок при роботі з файлом, якщо він не існує або недоступний.
9. **Відмінності текстового і двійкового потоку**Текстовий потік обробляє дані як послідовність символів з можливістю перетворення (наприклад, кодування, кінці рядків), а двійковий — працює з сирими байтами без форматування, що дозволяє зберігати точні двійкові дані.
10. **Режими відкриття файлових потоків у C++**Основні режими: читання, запис, додавання, бінарний режим, режим відкриття з очищенням файлу, відкриття без створення нового файлу.
11. **Класи файлових потоків у fstream**У бібліотеці fstream реалізовані класи для читання (ifstream), запису (ofstream) та читання-запису (fstream). Вони створюють відповідні об’єкти для роботи з файлами.
12. **Константи режимів відкриття файлових потоків в ios та їх призначення**Наприклад, ios::in — відкриття для читання, ios::out — для запису, ios::app — додавання в кінець файлу, ios::binary — двійковий режим, ios::trunc — очищення файлу при відкритті.
13. **Функції-члени для відкриття, закриття і перевірки кінця файлу**Використовуються методи open(), close(), а для перевірки кінця файлу — eof().
14. **Алгоритм видалення послідовності символів із текстового файла**Прочитати файл, зберегти вміст у тимчасовий буфер, видалити задану послідовність символів у буфері, відкрити файл для запису, записати оновлений текст назад, закрити файл.
15. **Складові типи даних у C/C++ та особливості їх оголошення**Складові типи — це структури, об’єднання, масиви, переліки. Вони містять кілька значень (елементів) різного або однакового типу, на відміну від простих типів (int, char тощо). Оголошення передбачає опис структури і її елементів.
16. **Перевірка, додавання, вилучення елемента у множині у C/C++**Перевірка здійснюється пошуком елемента, додавання — вставкою нового, вилучення — видаленням з контейнера (наприклад, std::set). Залежить від конкретної реалізації множини.
17. **Явне та неявне перетворення типів у C/C++**Неявне — компілятор автоматично конвертує типи там, де це можливо (наприклад, int у double). Явне — програміст використовує оператори приведення типів (static\_cast, (type)), щоб уникнути неоднозначностей.
18. **Доцільність використання union та приклад**union корисний, коли потрібно зекономити пам’ять і використовувати одну ділянку пам’яті для різних типів даних, але не одночасно. Наприклад, для збереження різних форматів даних, що використовуються по черзі.
19. **Алгоритм перевірки наявності заданого натурального числа у текстовому файлі:** Відкрити файл для читання, посимвольно або пословно зчитувати дані, порівнювати з шуканим числом, у разі знаходження — завершити пошук, закрити файл.
20. **Значення змінної symbol\_transaction при short symbol\_transaction = sizeof short('R');** Вираз sizeof визначає розмір типу short, а не самого символу. Тому змінна отримає розмір типу short (зазвичай 2 байти). Аргумент 'R' не впливає на розмір.