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

Балан М. С.

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** ModulesBalan

**ПОСТАНОВА ЗАДАЧІ**

Необхідно розробити модуль для електронного довідника пільг, який забезпечує виведення всіх записів на екран із перевіркою коректності даних та експорт усіх записів у текстовий файл із заданою назвою для зручного зберігання та перегляду.

**АНАЛІЗ ЗАДАЧІ**

Для реалізації модуля потрібно створити функцію виведення, яка перебирає вектор записів, відображає всі поля структури Benefit на екран, замінюючи порожні значення на "Немає" для зрозумілості, і функцію експорту, яка записує всі записи у текстовий файл у зрозумілому форматі, пропускаючи некоректні записи та перевіряючи успішність операції, при цьому підтримуючи коректну обробку кириличних символів.

**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

**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** ModulesSavelyev

**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

**ВИСНОВОК**

1. Масиви забезпечують ефективне зберігання однотипних елементів.
2. Структури зручні для моделювання об'єктів із кількома характеристиками.
3. Об'єднання (union) дозволяє економити пам’ять, зберігаючи лише одне з можливих полів.
4. Переліки (enum) роблять код зрозумілішим при роботі з фіксованими значеннями.
5. Множини реалізуються через побітові поля або контейнери STL.
6. Типи даних можуть автоматично або вручну змінюватися один в один.
7. Для роботи з файлами в C++ використовують бібліотеку <fstream>.
8. Клас ifstream застосовується для зчитування з файлів.
9. ofstream — для запису в файли.
10. fstream дозволяє як читати, так і записувати.
11. Символи й текст зручно обробляти за допомогою типів string та масивів char.
12. Клас string має широкий набір засобів для роботи з текстом.
13. Функції з <cstring> працюють із C-рядками.
14. Перед роботою з файлами потрібно перевіряти, чи відкрито файл.
15. Файли можуть містити як текстові, так і бінарні дані.
16. Бінарні файли займають менше місця, проте їх складніше читати вручну.
17. Завершивши роботу, файли потрібно обов’язково закривати.
18. Структури можна записувати й зчитувати у вигляді блоків у бінарному форматі.
19. Для зчитування символів та рядків використовують get() і getline().
20. Ефективна обробка інформації передбачає використання функцій.
21. Структури передаються у функції як за значенням, так і за посиланням.
22. Масиви передаються у функції у вигляді вказівників.
23. Для роботи з масивами необхідно знати їх розмір.
24. Константи допомагають уникнути помилок, задаючи фіксовані розміри масивів.
25. Перевірка кінця файлу (eof()) є обов’язковою під час читання.
26. seekg() та seekp() дозволяють змінювати позицію у файлі.
27. Масиви структур — зручна форма зберігання складних даних.
28. У структури можна вкладати інші структури.
29. Для символьних полів у структурах доцільно використовувати масиви char.
30. Перевірка введених даних допомагає уникнути помилок.
31. Модулі рекомендується розділяти на заголовочні (.h) і реалізаційні (.cpp) файли.
32. Інкапсуляція сприяє структурованості програм.
33. Власні типи роблять програмний код більш зрозумілим.
34. Імена змінних повинні чітко відображати їх призначення.
35. Під час зчитування можна одразу формувати масиви або списки.
36. Коментарі допомагають підтримувати та розуміти код.
37. Використання switch із enum спрощує обробку варіантів.
38. Множини зберігають тільки унікальні значення.
39. Контейнери STL (vector, set, map) розширюють можливості обробки даних.
40. Операції з файлами можуть бути буферизованими для підвищення продуктивності.
41. При ручному управлінні пам’яттю слід бути дуже обережним.
42. Дані з файлів можна аналізувати й фільтрувати.
43. Програми мають обробляти помилки вводу для підвищення надійності.
44. У Code::Blocks зручно працювати з багатофайловими проєктами.
45. Структури можна серіалізувати для збереження у файл.
46. typedef або using допомагають створювати синоніми для типів.
47. Дати доцільно зберігати у вигляді трьох полів: день, місяць, рік.
48. Дані можна сортувати ще під час їх зчитування.
49. Модулі мають бути чітко поділені за функціональністю.
50. Кожен модуль має реалізовувати окреме завдання.
51. При відкритті файлів потрібно обов’язково перевіряти можливі помилки.
52. Метод is\_open() дозволяє переконатися, що файл відкрито.
53. Для числових даних підходить оператор >>.
54. Рядки доцільно зчитувати за допомогою getline().
55. Дані можна зберігати у зручному для користувача вигляді.
56. Формат CSV зручний для подальшого аналізу.
57. Перевірка унікальності записів за ключовим полем є важливою.
58. Розбиття коду на функції підвищує його структурованість.
59. Коментарі є невід’ємною частиною якісного програмного коду.
60. Валідація введених даних забезпечує стабільність роботи.
61. Використання констант допомагає уникати помилок.
62. Індексація масивів починається з нуля.
63. Для сортування можна використовувати різні алгоритми: вибірки, вставки, бульбашки.
64. Під час читання файлів можна використовувати тимчасові буфери.
65. При роботі з текстом потрібно враховувати його кодування.
66. Код має бути зрозумілим і працювати на різних системах.
67. Структури доцільно використовувати для логічно пов’язаних даних.
68. Перед обробкою зчитаних даних їх потрібно перевіряти.
69. Нормалізовані дані полегшують подальшу обробку.
70. Модульна структура проєкту сприяє повторному використанню коду.
71. Усі відкриті файли необхідно закривати після завершення роботи.
72. Буферизація корисна при роботі з великими обсягами даних.
73. Структури зручно передавати у функції для зміни їх вмісту.
74. Формат помилкових рядків під час зчитування слід обробляти окремо.
75. Знання роботи з файлами та типами даних — основа створення складних програм.

**ВІДПОВІДЬНА КОНТРОЛЬНІ ЗАПИТАННЯ**

**1. Призначення та синтаксис try-throw-catch у C++**  
Конструкція try-throw-catch застосовується для обробки винятків, тобто помилок, які можуть виникнути під час виконання програми. У блоці try розміщується код, де може статись помилка. Ключове слово throw використовується для генерації винятку. Блок catch перехоплює цей виняток і дозволяє безпечно його обробити.

**2. Приклад міжмодульної змінної**  
Міжмодульна змінна — це змінна, доступна в різних частинах програми. Її оголошують у заголовочному файлі з модифікатором extern, а визначають у одному з .cpp файлів. Це дозволяє ділитися змінною між модулями.

**3. Область видимості змінних у функції main**  
Усі об'єкти, створені в межах функції main, мають локальну область видимості — вони доступні лише в середині цієї функції та не видно зовні.

**4. Порівняння enum і масиву**  
enum — це набір іменованих цілих констант, що підвищує зрозумілість коду. Масив — це структура, що дозволяє зберігати кілька значень одного типу під індексами.

**5. Потік і файловий потік**  
Потік — це канал передачі даних. Файловий потік — це тип потоку, який працює з файлами, на відміну від стандартного вводу/виводу.

**6. Алгоритм роботи з файлами через потоки**  
Спочатку відкривається файл у необхідному режимі. Далі виконується читання або запис, із перевіркою на успішне відкриття та завершення файлу. Після завершення файл обов'язково закривається.

**7. Обсяг текстового файлу з числом π до 106 знаків після коми**  
Число матиме 1 символ до коми, 1 символ коми, 106 цифр після коми — разом 108 символів. Якщо один символ = 1 байт, то обсяг ≈ 108 байт.

**8. Як перевірити, чи відкрився файл**  
Для цього використовується функція is\_open(). Вона дозволяє переконатися, що файл відкрився правильно і з ним можна працювати.

**9. Відмінність текстового і бінарного потоку**  
Текстовий потік працює з символами і може виконувати перетворення (наприклад, кінець рядка). Бінарний потік обробляє дані в сирому вигляді, без змін, що робить його ефективнішим, але менш зручним для перегляду.

**10. Основні режими відкриття файлів у C++**  
Файли можна відкривати для читання (in), запису (out), додавання (app), у бінарному вигляді (binary) або з очищенням вмісту (trunc).

**11. Класи файлових потоків у <fstream>**

* ifstream — для читання,
* ofstream — для запису,
* fstream — для обох операцій одночасно.

**12. Константи режимів відкриття у ios**

* ios::in — читання,
* ios::out — запис,
* ios::app — дописування в кінець,
* ios::binary — бінарний режим,
* ios::trunc — видалення вмісту при відкритті.

**13. Основні методи файлових потоків**

* Відкриття: open(),
* Закриття: close(),
* Перевірка кінця файлу: eof().

**14. Як видалити послідовність символів із файла**  
Прочитати вміст у буфер або тимчасовий файл, пропустити непотрібні символи, зберегти очищений вміст назад у файл.

**15. Складені типи даних і їх обробка**  
До таких типів належать масиви, структури, об’єднання, enum. Їх обробка залежить від доступу до елементів, обсягу пам’яті та способу зберігання.

**16. Робота з множинами: пошук, додавання, вилучення**  
Множини можна реалізовувати як побітові масиви або через STL-контейнери. Додавання — установка біта, видалення — обнулення, пошук — перевірка значення біта.

**17. Перетворення типів у C++**

* Неявне — компілятор автоматично змінює тип (наприклад, int → double).
* Явне — програміст використовує кастинг (наприклад, (int)x).

**18. Коли доцільно використовувати union**  
Коли потрібно заощадити пам’ять, зберігаючи у структурі лише одне з кількох можливих значень одночасно.

**19. Як знайти число у текстовому файлі**  
Зчитувати дані з файлу (по символу або слову) й порівнювати з цільовим числом. Якщо знайдено — виконати необхідну дію.

**20. Що зберігається у symbol\_transaction = sizeof short('R')**  
Оператор sizeof повертає розмір типу short, незалежно від значення 'R'. Зазвичай це 2 байти, і саме це значення зберігається у змінній