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

Федотов А. О.

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** ModulesFedotov

**ПОСТАНОВА ЗАДАЧІ**

Необхідно розробити модуль для роботи з електронним довідником пільг, який забезпечує збереження всіх записів у бінарний файл для подальшого використання та пошук запису за введеним кодом пільги з виведенням його даних на екран, із підтримкою коректної обробки помилок і валідацією даних.

**АНАЛІЗ ЗАДАЧІ**

Для реалізації модуля потрібно створити функцію збереження, яка записує вектор записів у бінарний файл у форматі benefit\_data.bin, ігноруючи некоректні записи та перевіряючи успішність операції, а також функцію пошуку, яка переглядає вектор записів, знаходить перший збіг за кодом пільги, виводить його дані або повідомлення про відсутність запису, забезпечуючи при цьому коректну роботу з кирилицею та захист від некоректних даних.

**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

**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. Робота з масивами дозволяє ефективно зберігати однотипні дані.
2. Структури зручно застосовувати для опису об'єктів із кількома властивостями.
3. Об'єднання (union) дозволяє економити пам’ять, зберігаючи лише один із можливих типів.
4. Переліки (enum) роблять код читабельнішим при роботі з фіксованими значеннями.
5. Множини реалізуються як побітові поля або STL-контейнери.
6. Типи даних можна перетворювати явно або неявно.
7. Файловий ввід/вивід у C++ забезпечується через бібліотеку <fstream>.
8. ifstream використовується для читання з файлу.
9. ofstream використовується для запису у файл.
10. fstream дозволяє і читати, і писати у файл.
11. Символьні дані зручно обробляти як рядки (string, char[]).
12. Стандартна бібліотека string має потужний функціонал для роботи з текстом.
13. Функції бібліотеки <cstring> працюють з C-рядками.
14. Для роботи з файлами важливо перевіряти, чи відкрився файл.
15. Дані у файлі можуть зберігатись у текстовому або бінарному форматі.
16. Бінарний формат займає менше місця, але його важче читати вручну.
17. Важливо закривати файл після завершення роботи з ним.
18. Структури можна зчитувати й записувати як блоки в бінарні файли.
19. Символи можна зчитувати по одному (get()), рядки — за допомогою getline().
20. Для ефективної обробки даних доцільно використовувати функції.
21. Передача структур у функції може відбуватись за значенням або за посиланням.
22. Масиви передаються у функції як вказівники.
23. Операції над масивами потребують знання їх розміру.
24. Для уникнення помилок корисно використовувати константи для розміру масивів.
25. При роботі з файлами слід перевіряти кінець файлу (eof()).
26. seekg() і seekp() дозволяють переміщатись у файлі.
27. Масив структур — це ефективний спосіб зберігати дані.
28. Структури можна вкладати одна в одну.
29. Для зберігання символьних полів структури можна використовувати масиви char.
30. Перевірка введення захищає від некоректних даних.
31. Модулі зручно розбивати на .h і .cpp файли.
32. Інкапсуляція допомагає структурувати програму.
33. Користувацькі типи покращують читабельність коду.
34. Імена змінних мають бути інформативними.
35. Під час зчитування з файлу можна формувати масиви або списки.
36. Код має бути коментований для зручності.
37. switch з enum полегшує обробку вибору.
38. Множини дозволяють зберігати унікальні значення.
39. STL-контейнери (vector, set, map) можуть бути корисними.
40. Операції введення/виведення можуть бути буферизованими.
41. Ручне управління пам’яттю вимагає обережності.
42. Дані з файлу можна аналізувати та фільтрувати.
43. Програми мають бути адаптовані до помилок вводу.
44. У Code::Blocks зручно використовувати багатофайлові проєкти.
45. Структури можна серіалізувати у файл.
46. typedef або using допомагає створювати синоніми типів.
47. Можна створити структуру з датою, використовуючи три поля: день, місяць, рік.
48. Дані можна сортувати при зчитуванні з файлу.
49. Модулі мають бути ізольовані за функціоналом.
50. Кожен модуль повинен мати чітке призначення.
51. Важливо обробляти помилки відкриття або читання файлу.
52. is\_open() дозволяє перевірити стан файлу.
53. Для введення чисел із файлу можна використовувати >>.
54. Для рядків — getline().
55. Можна писати дані у файл у зручному для користувача форматі.
56. Або у форматі CSV — для подальшого аналізу.
57. Можна перевіряти унікальність даних за ключем.
58. Розділення коду на функції підвищує його модульність.
59. Коментарі — важлива частина підтримки коду.
60. Валідація введення — необхідна умова стабільної програми.
61. Використання константних значень запобігає помилкам.
62. Індексація в масивах починається з нуля.
63. Для сортування можна використати алгоритми: вибірки, вставки, бульбашки.
64. Дані з файлу можна зберігати у тимчасовий буфер.
65. Для обробки символьних даних слід враховувати кодування.
66. Код має бути портативним і легко читаним.
67. Використання структури struct доречне для логічно пов’язаних змінних.
68. Усі дані з файлу мають бути перевірені перед обробкою.
69. Зручніше працювати з даними, коли вони нормалізовані.
70. Модульність дозволяє перевикористовувати код.
71. Усі відкриті файли мають бути закриті після роботи.
72. Під час читання великих файлів доцільно використовувати буферизацію.
73. Структури зручно передавати між функціями для зміни їх вмісту.
74. При читанні з файлу важливо обробляти можливість помилки формату.
75. Робота з типами даних і файлами — фундамент для створення серйозних додатків.

**ВІДПОВІДЬНА КОНТРОЛЬНІ ЗАПИТАННЯ**

1. **Призначення та синтаксис блоку try-throw-catch у C++**  
   Цей блок використовується для обробки виняткових ситуацій (помилок) під час виконання програми.

try містить код, де може виникнути помилка.

throw використовується для "викидання" винятку, коли виникає помилка.

catch перехоплює і обробляє викинутий виняток, щоб уникнути аварійного завершення програми.

1. **Приклад опису і використання міжмодульної змінної**  
   Міжмодульна змінна — це змінна, доступна у кількох файлах програми. Для цього вона оголошується у заголовочному файлі з ключовим словом extern і визначається в одному з .cpp файлів. Це дозволяє розділяти дані між модулями.
2. **Область видимості об’єктів, описаних у тілі функції main**  
   Змінні, типи, константи, описані у функції main, мають локальну область видимості. Вони доступні лише всередині цієї функції і недоступні з інших частин програми.
3. **Порівняння enum і масиву**

enum — це перелік констант з іменами, які зручно використовувати для покращення читабельності коду. Значення є цілими і унікальними.

Масив — це послідовність однотипних елементів, які індексуються числом, і служить для зберігання множини даних.

1. **Визначення потоку і файлового потоку**  
   Потік — це послідовність даних, яка передається або приймається у програмі. Файловий потік — це спеціальний потік для роботи з файлами на диску. Він відрізняється тим, що оперує файлами, а не стандартним вводом/виводом (екран, клавіатура).
2. **Універсальний алгоритм читання/запису даних у файл за допомогою файлового потоку**  
   Спочатку відкривається файл у відповідному режимі (читання або запис). Потім послідовно зчитуються або записуються дані, при цьому перевіряється чи файл відкрився успішно і чи не досягнуто кінець файлу. По завершенню файл закривається.
3. **Об’єм текстового файлу з числом π з точністю 106 знаків після коми**  
   Об’єм приблизно дорівнює кількості символів, включно з цілою частиною, десятковою крапкою і знаками після коми. Для 106 знаків після коми це близько 108 символів (1 + 1 + 106), кожен символ — 1 байт, отже ~108 байт.
4. **Перевірка статусу відкриття файлового потоку**  
   Перевірка полягає у виклику спеціальної функції або методу, який повертає, чи файл відкрився успішно. Це потрібно для уникнення роботи з неіснуючими або недоступними файлами, що запобігає помилкам.
5. **Різниця між текстовим і двійковим потоками**  
   Текстовий потік працює з символами у вигляді тексту (людськочитаний формат), може виконувати конвертацію символів (наприклад, кодування, перетворення кінців рядків). Двійковий потік працює з сирими байтами без змін, тому він ефективніший для зберігання будь-яких даних.
6. **Режими відкриття файлових потоків у C++**  
   Основні режими: читання, запис, додавання (дописування), бінарний режим, режим створення (створити новий файл), режим відкриття існуючого файлу.
7. **Класи файлових потоків у <fstream>**  
   ifstream — для читання з файлу,  
   ofstream — для запису у файл,  
   fstream — для одночасного читання і запису.  
   При створенні об’єкта можна відразу відкрити файл.
8. **Константи режимів відкриття в ios**  
   ios::in — відкрити для читання,  
   ios::out — для запису,  
   ios::app — додавання в кінець,  
   ios::binary — бінарний режим,  
   ios::trunc — обрізати файл при відкритті.
9. **Функції-члени для відкриття, закриття і перевірки EOF**  
   Для відкриття — open(),  
   для закриття — close(),  
   для перевірки кінця файлу — eof().
10. **Алгоритм видалення послідовності символів з текстового файла**  
    Зчитати файл у тимчасовий буфер або інший файл, пропускаючи задану послідовність символів, потім переписати результат назад.
11. **Складові типи даних у C/C++ і особливості їх обробки**  
    Масиви, структури, об’єднання, переліки — складаються з кількох елементів або типів. Для їх обробки потрібно враховувати розмір, індексацію (для масивів), доступ до полів (для структур).
12. **Перевірка наявності елемента у множині, додавання, вилучення**  
    Множини реалізують як побітові масиви або контейнери, де можна перевірити наявність за індексом, додати елемент шляхом встановлення біта, видалити — скинути.
13. **Явне і неявне перетворення типів у C++**  
    Неявне — коли компілятор автоматично конвертує тип (наприклад, int у double).  
    Явне — коли програміст спеціально вказує, що потрібно конвертувати (кастинг).
14. **Випадки доцільного використання union**  
    Якщо потрібно зберігати різні типи даних у одній і тій же області пам’яті, наприклад, різні типи параметрів, економлячи пам’ять.
15. **Алгоритм перевірки наявності заданого числа у текстовому файлі**  
    Зчитати посимвольно або пословно дані з файлу і порівнювати з заданим числом, якщо знайдено — повідомити.
16. **Що міститиме змінна symbol\_transaction = sizeof short('R')?**  
    Оператор sizeof повертає розмір типу short в байтах. Аргумент 'R' — символ, але sizeof оцінює тип, а не значення, тому результат — це розмір типу short (зазвичай 2 байти). Змінна містить це значення.