ПРОГА 3 СЕМ (Жереб)

1. Що таке клас? Що таке об'єкт? В чому між ними різниця? Які сутності предметної області описуються класами, а які – об'єктами? Чи всі сутності предметної області мають описуватись класами чи об'єктами? (Навести приклади з власного коду)

2. Як правильно структурувати код програми, щоб розділяти код, що описує логіку програми, та код інтерфейсу користувача? (Навести приклади з власного коду)

3. Що таке поля (fields), властивості(properties), методи (methods), аксесори (accessors/getters/setters)? В чому між ними різниця? Коли варто використовувати кожен з них? (Навести приклади з власного коду)
3.1. Що таке змінні (variables),...(далі те саме)
3.2 …,функції (functions),...
3.3 …, атрибути (attributes),....

4. Що таке події (events) в інтерфейсі користувача? Які події можуть виникати під час роботи користувача з програмою? Як правильно реагувати на них? (Навести приклади з власного коду)

5. Що таке інкапсуляція (encapsulation)? Що таке рівні доступу public, protected, private? В чому між ними різниця? Коли варто використовувати кожен з них? (Навести приклади з власного коду)
5.1 Що таке інкапсуляція (encapsulation)? Які бувають різновиди інкапсуляції? Що таке рівні доступу public, protected, private? В чому між ними різниця? Коли варто використовувати кожен з них? (Навести приклади з власного коду)

6. Як перевіряти правильність даних, які вводить користувач? (Навести приклади з власного коду)

7. Що таке успадкування (inheritance)? Як пов'язані між собою базовий та успадкований класи? Які елементи базового класу доступні з успадкованого, і навпаки?

7.1Коли доцільно використовувати успадкування? (Навести приклади з власного коду)
7.2 В чому відмінність між public та private успадкуванням? (Навести приклади з власного коду)

8. Якщо є набір даних, і кожен елемент даних містить декілька властивостей - як сортувати дані за однією чи кількома з цих властивостей? Як показувати відсортовані дані користувачу? Як надати можливість користувачу обрати, за якими властивостями сортувати? Як показати, за якими властивостями зараз відсортовані дані? (Навести приклади з власного коду)

9. Що таке поліморфізм (polymorphism)? До яких елементів класу він застосовується? Коли доцільно використовувати поліморфізм? Що означає модифікатор override і коли доцільно його використовувати? (Навести приклади з власного коду)
9.1 В чому відмінність між статичним та динамічним поліморфізмом? (Навести приклади з власного коду)

10. Як можна надати користувачу можливість вибирати певні елементи даних і залежно від цього фільтрувати набори даних, так щоб виводити лише необхідні дані? Як реалізувати пошук в наборі даних за певними критеріями? Як показати користувачу лише необхідні дані, замість повного набору даних? (Навести приклади з власного коду)

11. Що таке віртуальні (virtual) елементи класу? Які елементи класу можуть бути віртуальними? Як віртуальні елементи пов'язані з інкапсуляцією, успадкуванням, поліморфізмом? (Навести приклади з власного коду)

12. Як можна надати користувачу можливість вибирати певні елементи даних і залежно від цього показувати різну інформацію, яка стосується обраного елемента даних? (Навести приклади з власного коду)

13. Що таке абстрактні (abstract) класи? Що таке чисто віртуальні (pure virtual) методи? Як описати абстрактні класи та чисто віртуальні методи в мові програмування? Коли їх варто використовувати? В чому відмінність між

абстрактним класом та інтерфейсом (interface)? (Навести приклади з власного коду)

14. Якщо в програмі зберігається забагато інформації, і одразу показувати користувачу всю інформацію недоцільно – як організувати інтерфейс користувача, щоб зручним чином надати доступ до всієї інформації і при цьому не перевантажувати користувача даними? (Навести приклади з власного коду).

15. Що таке конструктор? Для чого використовуються конструктори? Чи всі класи мають містити конструктори? Чи може клас містити декілька конструкторів? Чи можуть конструктори бути віртуальними, і якщо так - коли це використовується? Коли і в якому порядку викликаються конструктори? (Навести приклади з власного коду)

16. Що таке деструктор? Для чого використовуються деструктори? Чи всі класи мають містити деструктори? Чи може клас містити декілька деструкторів? Чи можуть деструктори бути віртуальними, і якщо так - коли це використовується? Коли і в якому порядку викликаються деструктори? (Навести приклади з власного коду)

17. Що таке модифікатор const? Де він використовується і що означає? Коли доцільно його використовувати? Чи зміниться поведінка програми, якщо його прибрати? (Навести приклади з власного коду)

18. Що таке керування пам'яттю (memory management)? Що таке втрати пам'яті (memory leaks)? Як правильно створювати та знищувати об'єкти, щоб не виникало проблем? (Навести приклади з власного коду)

19. Що таке "розумні вказівники" (smart pointers)? Для чого їх використовувати? Які переваги та недоліки їх використання, порівняно зі звичайними вказівниками (raw pointers), посиланнями (references) та просто екземплярами об'єктів? (Навести приклади з власного коду).
19.1 Що таке "розумні вказівники" (smart pointers)? Які бувають різновиди розумних вказівників?....(все те саме)

1. Що таке клас? Що таке об'єкт? В чому між ними різниця? Які сутності предметної області описуються класами, а які – об'єктами? Чи всі сутності предметної області мають описуватись класами чи об'єктами? (Навести приклади з власного коду)

Клас- це типи даних, які описують однакові чи схожі за властивостями та поведінкою об’єкти і використовуються для групування змінних та функцій. В ООП змінні класу називаються полями (fields), функції класу-методи(methods).
Об’єкт - створений та ініціалізований екземпляр класу називають об’єктом класу. На основі одного класу, може бути створено безліч об’єктів, які відрізнятимуться один від одного своїм значеннями полів.

-Описом об’єкта є клас, а об’єкт є екземпляр цього класу.-

Клас це певний шаблон, що описує структуру і поведінку, визначає які властивості (поля) і методи можуть бути у об’єкта. У заагальному вигляді не виконує дії, лише описує їх. Об’єкт в свою чергу це конкретний екземпляр класу. Він містить конкретні значення полів, атрибутів визначених класом, виконує дії за допомогою методів класу.
Об’єкти, як екземпляри класів, є динамічними структурами, тобто вони створюються і знищуються в процесі виконання програми- під нього виділяється пам’ять. Кожен об’єкт класу має власну копію усіх властивостей класу. На противагу об’єктам, класи є статичними структурами. Усі їх властивості є наперед (на етапі компіляції) визначеними.

Класи використовують для складних сутностей із поведінкою або властивостями, які часто повторюються в системі. Простішим сутностям (числам, рядкам, спискам) достатньо примітивних типів або структур даних. Використання класів виправдане, коли це спрощує розробку, повторне використання та підтримку коду.

2. Як правильно структурувати код програми, щоб розділяти код, що описує логіку програми, та код інтерфейсу користувача? (Навести приклади з власного коду)(full GPT, хз шо писать ще, але те що є норм)

- В першу чергу слід розділити код на модулі, які відповідатимуть за конкретні аспекти програми: логіку бізнес-процесів, доступ до даних, інтерфейс користувача, тощо.
- Також обов'язково потрібно створити окремі класи або файли для кожної функціональної частини програми.
- Хороши рішенням буде створити інтерфейсів або абстрактних класів.
- Принцип розділення відповідальності (Single Responsibility Principle, SRP), щоб кожен клас або модуль відповідав тільки за одну конкретну задачу також допоможе в вирішенні проблеми.
- Також можна організувати файли та класи в різні папки, щоб логіка бізнес-процесів знаходилася в окремих папках, а інтерфейс користувача — в інших.
- Дотримання принципу DRY (Don’t Repeat Yourself), тобто уникнення дублювання коду, створюючи функції або методи для повторного використання логіки програми.
- Використовувати архітектурні патерни

3. Що таке поля (fields), властивості(properties), методи (methods), аксесори (accessors/getters/setters)? В чому між ними різниця? Коли варто використовувати кожен з них? (Навести приклади з власного коду)
3.1. Що таке змінні (variables),...(далі те саме)
3.2 …,функції (functions),...
3.3 …, атрибути (attributes),....

Клас може зберігати деякі дані. Для зберігання даних у класі застосовуються поля (fields). Поле класу або атрибут в об'єктно-орієнтованому програмуванні — змінна, зв'язана з класом або об'єктом. Всі дані об'єкта зберігаються в його полях. Доступ до полів здійснюється по імені. Зазвичай тип даних кожного поля задається в описі класу, членом якого є поле.
Крім того, клас може визначати деяку поведінку або дії, що виконуються. Для визначення поведінки у класі застосовуються методи(methods). Метод – іменований блок коду, який виконує деякі дії, які впливають лише на один об‘єкт. Якщо змінні зберігають деякі значення, то методи містять набір інструкцій, які виконують певні дії.

(В чому різниця методів і функцій? Функції (functions) (такі ж підпрограми) — це відносно самодостатні, відносно незалежні частини коду, які складають більшу програму.)

Властивості (properties) в розробці програмного забезпечення стосуються атрибутів (attributes) або даних, які містять об’єкти в об’єктно-орієнтованому програмуванні (ООП). Вони використовуються для зберігання інформації про об’єкт, і, як правило, їх можна як читати, так і записувати, хоча деякі властивості можуть бути лише для читання або лише для запису залежно від того, як вони реалізовані.
Властивості відіграють вирішальну роль в інкапсуляції, одному з основних принципів ООП. Вони забезпечують контрольований інтерфейс до даних об’єкта. Замість прямого доступу до даних об’єкта, що може призвести до неочікуваних помилок, властивості дозволяють розробникам визначати геттери (getters) та сетери (setters). Геттери повертають значення властивості, а сеттери перевіряють і призначають їм значення.
Геттери: методи, які отримують значення властивості.
Сетери: методи, які встановлюють або оновлюють значення властивості після виконання будь-якої необхідної перевірки.
Правильне використання властивостей підвищує зручність обслуговування, безпеку та надійність програмних додатків.

Accessor – функція, яка використовується для отримання даних захищених елементів. Вони використовуються замість того, щоб робити змінну елемента класу загальнодоступною та змінювати її безпосередньо в об’єкті. Щоб отримати доступ до приватного (private) елемента об’єкта, необхідно викликати функцію доступу (accessor). Зазвичай назва аксесору починається з префікса get.

Атрибути (attributes) є однією з ключових особливостей сучасного C++, яка дозволяє програмісту вказувати додаткову інформацію для компілятора, щоб застосувати обмеження (умови), оптимізувати певні фрагменти коду. Простіше кажучи, атрибут діє як анотація або примітка для компілятора, яка надає додаткову інформацію про код для цілей оптимізації та виконання певних умов щодо нього.
Отже, призначеннями атбирутів є:

1. Накладання обмеження на код: тут обмеження стосується умови, якій мають відповідати аргументи певної функції для її виконання (передумова).
2. Надати компілятору додаткової інформації для цілей оптимізації: компілятори дуже гарні в оптимізації, але порівняно з людьми вони все ще відстають у деяких місцях і пропонують узагальнений код, який є не дуже ефективним. В основному це відбувається через відсутність додаткової інформації про «проблему», яка є у людей.
3. Обхід певних попереджень і помилок, які програміст мав намір мати у своєму коді: це трапляється рідко, але іноді програміст навмисно намагається написати помилковий код, який виявляє компілятор і повідомляє про помилку або видає попередження. Одним із таких прикладів є невикористана змінна, яка була залишена в такому стані з певної причини, або оператор switch, де оператори break не використовуються в деяких випадках, щоб викликати умови "провалу". (Щоб обійти помилки та попередження про такі умови, C++ надає такі атрибути, як [maybe_unused] і [fallthrough], які запобігають створенню компілятором попереджень або помилок. В C++ існує список стандартних атрибутів.)

(!)4. Що таке події (events) в інтерфейсі користувача? Які події можуть виникати під час роботи користувача з програмою? Як правильно реагувати на них? (Навести приклади з власного коду) (GPT+Forums)

Події- дія яка розпізнається програмою та обробляється за допомогою певних інструкцій. У роботі з користувацьким інтерфейсом це певний механізм що повідомляє програму про певні дії або зміни, що можуть бути викликаними діями користувача або системними змінами.
Різновид подій під час роботи користувача з програмою:

- Події взаємодії з мишею (кліки, наведення, переміщення)
- Події клавіатури (натискання клавіші)
- Події фокусу (отримання або втрата фокусу)
- Події форми (зміна значень, сабміти та скидання)

(!)Для правильного реагування на події використовуються різні методи і умовах програмування. Це використання слухачів(event listeners), що прив'язуються до події і автоматично реагують викликаючи певні обробники. Делегування подій

5. Що таке інкапсуляція (encapsulation)? Що таке рівні доступу public, protected, private? В чому між ними різниця? Коли варто використовувати кожен з них? (Навести приклади з власного коду)
5.1 Що таке інкапсуляція (encapsulation)? Які бувають різновиди інкапсуляції? Що таке рівні доступу public, protected, private? В чому між ними різниця? Коли варто використовувати кожен з них? (Навести приклади з власного коду)

Інкапсуляція є одним із принципів ООП, що забезпечує приховування внутрішніх даних об'єктів класів для запобігання прямому доступу до них. Вона дозволяє захистити поля об'єкта і забезпечити контрольований доступ до його даних через спеціально визначені методи, такі як **гетери** (методи доступу) та **сетери** (методи встановлення). Завдяки цьому інші класи можуть взаємодіяти з даними лише відповідно до встановлених правил.

Рівні доступу:

- **public (загальнодоступний)**

Використовується для методів і змінних, які повинні бути доступними в будь-якому місці програми.

- **private(приватний)**

Модифікатор доступу, який використовується для обмеження видимості полів, методів і конструкторів визначальним класом, тобто елементи доступні тільки в межах самого класу. Цей механізм інкапсуляції допомагає захистити внутрішній стан об’єкта та сприяє приховуванню даних.

- **protected(захищений)**

Елементи модифікатору protected дозволяють забезпечити частковий контроль над доступом до даних, сприяючи інкапсуляції. Елементи з ци модифікатором доступні в межах класу, де вони були оголошені, а також у класах-нащадках, які успадковують цей клас. Це забезпечує доступність для похідних класів, але обмежує доступ для інших класів, роблячи ці елементи захищеними від прямого використання ззовні.

У чому різниця між ними?

Рівень доступу	Видимість з іншого класу	Видимість у спадкоємців
`public`	Так	Так
`protected`	Ні	Так
`private`	Ні	Ні

private: для внутрішніх даних і методів, які не повинні бути змінені або викликані безпосередньо.
protected: для методів і змінних, які можуть бути успадковані та змінені в класах-нащадках.
public: для методів і змінних, які є частиною зовнішнього інтерфейсу класу

Різновиди інкапсуляції:
Інкапсуляція змінних-членів: інкапсуляція даних, стосується змінних-членів класу. Для доступу приватних змінних використовуються методи-сеттери та геттери, які забезпечують контрольований доступ до даних.

Інкапсуляція функцій: Інкапсуляція функцій полягає в тому, щоб оголошувати методи коду як приватні, що запобігає доступу до них ззовні. Це дозволяє приховувати реалізацію логіки та захищати внутрішню поведінку класу від неавторизованого доступу.

Інкапсуляція класів: Інкапсуляція класів передбачає обмеження доступу до класу шляхом його оголошення як приватного. Це дозволяє приховати деталі реалізації класу та зробити його недоступним для інших класів. Такий підхід забезпечує контрольоване використання класу та сприяє підтриманню цілісності внутрішньої логіки програми.

6. Як перевіряти правильність даних, які вводить користувач? (Навести приклади з власного коду) (Варіант Артема)

Перевірка є важливою складовою для забезпечення безпеки та коректності програм. Існує кілька загальних підходів, які можна використовувати для перевірки введених даних:
- Використання умовних операторів
- Використання функції getline для рядкових вводів
- Обробка винятків (Exceptions)
- Використання вбудованих функцій мови програмування:
- Багато мов програмування мають вбудовані функції для перевірки типів даних, довжини рядків, числових діапазонів та інші. Наприклад, у Python можна використовувати isinstance(), len(), int(), float(), тощо.

Приклад Артема:
Зважаючи на те, що мій проєкт оформлений у формі гри, явного на прикладу на це питання теж немає. У рамках мого проєкту можна було б реалізувати можливість користувача вводити нікнейм та перевіряти його на коректність по таким параметрам як:
- словник символів
- довжина
- будь-які додаткові параметри
Після того як би користувач ввів свій нікнейм, певна функція б зчитувала рядок і за допомогою, наприклад, умовних операторів перевіряла б коректність такого вводу, у висновку підтверджуючи коректність вводу, або ж виводячи повідомлення про те, що ввід некоректний.

7. Що таке успадкування (inheritance)? Як пов'язані між собою базовий та успадкований класи? Які елементи базового класу доступні з успадкованого, і навпаки?
7.1Коли доцільно використовувати успадкування? (Навести приклади з власного коду)
7.2 В чому відмінність між public та private успадкуванням? (Навести приклади з власного коду)

Успадкування є однією з основних особливостей об’єктно-орієнтованого програмування. Здатність класу отримувати властивості та характеристики від іншого класу та повторно використовувати код, посилаючись на поведінку та дані об’єкта називається успадкуванням (inheritance). Іншими словами, клас, який успадковує від іншого класу, спільно використовує всі атрибути та методи класу, на який посилається.

Успадкований клас називається підкласом або дочірнім класом класу, від якого він успадковується. А клас, який успадковується, називається батьківським або базовим класом.

(GPT)

Ось основні випадки, коли успадкування доцільне:

Коли між класами існує логічний зв'язок типу "є-насправді" (is-a)

- Наприклад:

cpp Copy code

```cpp
class Animal {
    void eat() { /* ... */ }
};

class Dog : public Animal {
    void bark() { /* ... */ }
};
```

- **Пояснення:** Собака (*Dog*) є твариною (*Animal*), тому доцільно використовувати успадкування.

Повторне використання коду базового класу

- Наприклад:

```cpp
class Shape {
    void draw() { /* загальна логіка малювання */ }
};

class Circle : public Shape {
    void draw() { /* спеціалізована логіка */ }
};

class Rectangle : public Shape {
    void draw() { /* спеціалізована логіка */ }
};
```

  - **Пояснення:** Усі фігури мають метод `draw`, тому його можна визначити в базовому класі, а специфічні деталі реалізувати у дочірніх.

Поліморфізм

Успадкування є основою для досягнення поліморфізму, що дозволяє використовувати базовий клас як інтерфейс для дочірніх класів.

- Наприклад:

```cpp
class Shape {
public:
    virtual void draw() = 0; // Чисто віртуальна функція
};

class Circle : public Shape {
    void draw() override { /* малювання кола */ }
};

class Rectangle : public Shape {
    void draw() override { /* малювання прямокутника */ }
};

void render(Shape& shape) {
    shape.draw(); // Викликає відповідний метод залежно від типу об'єкта
}
```

  - **Пояснення:** Це дозволяє створювати гнучкий код, який може працювати з будь-яким підтипом базового класу.

Успадкування дозволяє винести загальний функціонал у базовий клас, що робить код менш дубльованим і простішим у підтримці.

- Наприклад:

cpp Copy code

```cpp
class Employee {
    std::string name;
    double salary;
    void calculatePay() { /* логіка для всіх співробітників */ }
};

class Manager : public Employee {
    void manageTeam() { /* специфічна логіка */ }
};

class Engineer : public Employee {
    void developCode() { /* специфічна логіка */ }
};
```

(GPT)

Базовий та успадкований класи пов'язані між собою через механізм успадкування, який дозволяє дочірньому (успадкованому) класу використовувати члени базового класу. Рівень доступності цих членів залежить від модифікаторів доступу (public, protected, private) та типу успадкування (public, protected, private).

Успадкований клас не впливає на доступ до базового класу. Базовий клас не має доступу до жодних членів дочірнього класу, навіть якщо вони є публічними. Доступ у зворотному напрямку не передбачений, оскільки базовий клас "не знає" про своїх нащадків.

-Модифікатори доступу та успадкування-

Більшість об'єктно-орієнтованих мов мають три форми модифікаторів доступу. Тож, хоча дочірній клас успадковує базовий клас, можна обмежити доступ дочірнього класу до певних методів і змінних у базовому класі. Також можна обмежити доступ до них, використовуючи код поза класом.

Далі GPT:

У контексті об'єктно-орієнтованого програмування (особливо в C++), **public** і **private** успадкування визначають, як члени базового класу стають доступними в дочірньому класі. Це впливає на рівень доступу до властивостей і методів базового класу.

При public успадкуванні всі **публічні** (public) члени базового класу залишаються публічними, а **захищені**(protected) члени залишаються захищеними в дочірньому класі.

При private успадкуванні всі члени базового класу (як public, так і protected) стають приватними (private) у дочірньому класі.

Основні відмінності:

Характеристика	Public успадкування	Private успадкування
Публічні члени базового класу	Залишаються публічними	Стають приватними
Захищені члени базового класу	Залишаються захищеними	Стають приватними
Використання поза класом	Доступні через дочірній клас	Недоступні через дочірній клас
Логіка	Дочірній клас розширює функціонал базового класу	Дочірній клас використовує базовий клас як частину своєї реалізації

8. Якщо є набір даних, і кожен елемент даних містить декілька властивостей - як сортувати дані за однією чи кількома з цих властивостей? Як показувати відсортовані дані користувачу? Як надати можливість користувачу обрати, за якими властивостями сортувати? Як показати, за якими властивостями зараз відсортовані дані? (Навести приклади з власного коду) (фул GPT, якась поєбєнь, в файлі з god bless чату краще написано)

Це наше:

1. Як сортувати дані за однією чи кількома властивостями

Сортування за однією властивістю

1. У мовах програмування зазвичай доступні стандартні функції для сортування (наприклад, `sort` у Python, C++, або JavaScript).
2. Використовуйте функцію-компаратор (custom comparator) для вказівки, за якою властивістю виконувати сортування.

   Приклад на Python:

```python
data = [{"name": "Alice", "age": 25}, {"name": "Bob", "age": 30}, {"name": "(
sorted_data = sorted(data, key=lambda x: x["age"])  # Сортуємо за віком
```

Сортування за кількома властивостями

1. У компараторі визначте порядок пріоритетів для властивостей.
2. У Python це можна зробити за допомогою `key` із кортежем значень:

   Приклад:

```python
data = [{"name": "Alice", "age": 25, "score": 90},
        {"name": "Bob", "age": 30, "score": 85},
        {"name": "Charlie", "age": 25, "score": 95}]
sorted_data = sorted(data, key=lambda x: (x["age"], -x["score"]))  # Спочатку
```

3. У мовах із функціями сортування (C++, Java) передавайте власні компаратори для сортування.

2. Як показувати відсортовані дані користувачу

1. **Таблиці:**
   - Відображайте дані у вигляді таблиці, де кожен стовпець відповідає властивості.
   - Використовуйте виділення (наприклад, стрілки ↑/↓) у заголовках стовпців, щоб показати напрямок сортування.

   Приклад:

   diff Copy code

```
Name      | Age  | Score
----------|------|-------
Charlie   | 25   | 95
Alice     | 25   | 90
Bob       | 30   | 85
```

2. **Графічний інтерфейс:**
   - У веб-додатках: використовуються таблиці (HTML + CSS), наприклад, з бібліотекою DataTables.
   - У десктоп-додатках: забезпечте можливість сортування, наприклад, кліком по заголовку стовпця.
3. **Фільтри й меню:**
   - Надайте випадаючий список або панель для вибору способу сортування.

3. Як надати можливість користувачу обрати, за якими властивостями сортувати

1. **Веб або графічний інтерфейс:**
   - Додайте інтерактивні елементи:
     - Списки з кількома варіантами (наприклад, "Сортувати за віком", "Сортувати за балом").
     - Кнопки або випадаючі меню для вибору пріоритету властивостей.
2. **Консольний інтерфейс:**
   - Запитуйте користувача через текстовий ввід:

```python
sort_key = input("Введіть властивість для сортування (name, age, score):
sorted_data = sorted(data, key=lambda x: x[sort_key])
```

3. **Динамічні комбінації:**
   - Дозвольте обрати кілька властивостей для сортування й визначити їх пріоритет.
   - Наприклад, через вибір "список властивостей" із зазначенням їх порядку.

4. Як показати, за якими властивостями зараз відсортовані дані

1. **Візуально (таблиці):**
   - У заголовках стовпців відображайте маркер сортування (наприклад, ↑/↓).
   - **Приклад:** Стовпець "Age ↑" вказує, що дані відсортовані за віком у зростаючому порядку.
2. **Текстовий вивід:**
   - Виводьте підказку перед таблицею:

```
Дані відсортовані за: age (зростання), score (спадання)
```

3. **Динамічний інтерфейс:**
   - У випадаючому меню або в панелі управління відображайте активні критерії сортування.

А це не наше (з godbless):

2. Якщо є набір даних, і кожен елемент даних містить декілька властивостей – як сортувати дані за однією чи кількома з цих властивостей? Як показувати відсортовані дані користувачу? Як надати можливість користувачу обрати, за якими властивостями сортувати? Як показати, за якими властивостями зараз відсортовані дані? (Приклад коду)

Відповідь:

- **Сортування за однією чи кількома властивостями:** Використовуємо стандартні алгоритми сортування, наприклад `std::sort`, із лямбда-функціями, які визначають, як порівнювати елементи.
- **Показ відсортованих даних:** Дані виводяться у зручному форматі (наприклад, таблицею).
- **Обрання властивостей для сортування:** Користувач вводить вибір, а програма виконує сортування відповідно до вибраної властивості.
- **Показ властивостей сортування:** Виводимо назву властивості, за якою виконано сортування.

Приклад коду:

cpp

Копировать код

```cpp
#include <iostream>
using namespace std;

// Структура даних
struct Item
{
    string name;
    int price;
    int quantity;
};

// Просте сортування методом бульбашки
void bubbleSort(Item items[], int size, int choice)
{
    for (int i = 0; i < size - 1; ++i)
    {
        for (int j = 0; j < size - i - 1; ++j)
        {
            bool condition = false;

            if (choice == 1) // Сортування за ім'ям
                condition = items[j].name > items[j + 1].name;
            else if (choice == 2) // Сортування за ціною
                condition = items[j].price > items[j + 1].price;
            else if (choice == 3) // Сортування за кількістю
                condition = items[j].quantity > items[j + 1].quantity;
```

cpp

Копировать код

```cpp
#include <iostream>
using namespace std;

// Структура даних
struct Item
{
    string name;
    int price;
    int quantity;
};

// Просте сортування методом бульбашки
void bubbleSort(Item items[], int size, int choice)
{
    for (int i = 0; i < size - 1; ++i)
    {
        for (int j = 0; j < size - i - 1; ++j)
        {
            bool condition = false;

            if (choice == 1) // Сортування за ім'ям
                condition = items[j].name > items[j + 1].name;
            else if (choice == 2) // Сортування за ціною
                condition = items[j].price > items[j + 1].price;
            else if (choice == 3) // Сортування за кількістю
                condition = items[j].quantity > items[j + 1].quantity;
```

```
    int choice;
    cout << "Choose sorting property:\n1. Name\n2. Price\n3. Quantity\n";
    cin >> choice;

    if (choice == 1)
    {
        bubbleSort(items, 3, choice);
        displayData(items, 3, "Name");
    }
    else if (choice == 2)
    {
        bubbleSort(items, 3, choice);
        displayData(items, 3, "Price");
    }
    else if (choice == 3)
    {
        bubbleSort(items, 3, choice);
        displayData(items, 3, "Quantity");
    }
    else
    {
        cout << "Invalid choice!" << endl;
    }

    return 0;
}
```

Пояснення:

- Замість `std::vector` використовується статичний масив `Item items[3]`.
- Для сортування застосовується метод бульбашки (`bubbleSort`), який працює з масивом.
- Програма підтримує сортування за трьома властивостями: ім'я, ціна, кількість.

Результат:

yaml — Копировать код

```
Choose sorting property:
1. Name
2. Price
3. Quantity
2
Sorted by: Price
Name: Banana, Price: 2, Quantity: 100
Name: Apple, Price: 3, Quantity: 50
Name: Cherry, Price: 5, Quantity: 30
```

9. Що таке поліморфізм (polymorphism)? До яких елементів класу він застосовується? Коли доцільно використовувати поліморфізм? Що означає модифікатор override і коли доцільно його використовувати? (Навести приклади з власного коду)
9.1 В чому відмінність між статичним та динамічним поліморфізмом? (Навести приклади з власного коду)

Поліморфізм — одна з основних концепцій об’єктно-орієнтованого програмування (ООП), яка описує, що можна отримати доступ до об’єктів різних типів через один і той самий інтерфейс. Кожен тип може забезпечити власну незалежну реалізацію цього інтерфейсу.

Отже, успадкування дозволяє нам успадковувати атрибути та методи з іншого класу, а поліморфізм використовує ці методи для виконання різних завдань. Це дозволяє нам виконувати одну дію різними способами.

(GPT)

До яких елементів класу застосовується поліморфізм?

1. **Методи**:
    - Поліморфізм здебільшого застосовується до методів класу.
    - Методи в базовому класі, позначені як `virtual` або `abstract`, можуть бути перевизначені в дочірньому класі.
2. **Інтерфейси**:
    - У мовах, що підтримують інтерфейси (наприклад, Java), поліморфізм дозволяє реалізувати різні варіанти методів для одного інтерфейсу.
3. **Оператори**:
    - Поліморфізм також застосовується до перевантаження операторів, це більше стосується статичного поліморфізму.

(GPT)

Коли доцільно використовувати поліморфізм?

1. **Коли потрібно реалізувати єдиний інтерфейс для різних типів об'єктів.**

    - Наприклад, у графічному редакторі всі фігури (коло, квадрат, трикутник) можуть викликати метод `draw`, незалежно від їхнього типу.
2. **Для розширюваності системи.**
    - Поліморфізм спрощує додавання нових типів, не змінюючи код, що працює з базовими класами чи інтерфейсами.
3. **Для спрощення коду.**
    - Можна створити універсальний алгоритм, який працює з базовим класом, а конкретна реалізація залежить від фактичного типу.

(GPT)

Що означає модифікатор `override` і коли доцільно його використовувати?

Модифікатор `override` використовується для позначення, що метод у дочірньому класі перевизначає (overrides) віртуальний (virtual) метод базового класу.

- **Значення:**
    - Гарантує, що метод дійсно перевизначає існуючий віртуальний метод із базового класу.
    - Якщо метод із модифікатором `override` не має відповідного методу в базовому класі, компілятор видасть помилку.

Поліморфізм під час компіляції (статичний) відноситься до механізму в C++, коли компілятор визначає, яку функцію викликати під час компіляції на основі сигнатури функції та контексту, в якому вона викликається. Зазвичай це досягається шляхом перевантаження функцій і операторів.

Поліморфізм під час виконання (динамічний) в C++, також відомий як пізнє зв’язування або динамічний поліморфізм, досягається за допомогою перевизначення функції (override). Це забезпечує більшу гнучкість, оскільки конкретна реалізація функції визначається на основі фактичного типу об’єкта під час виконання програми.

10. Як можна надати користувачу можливість вибирати певні елементи даних і залежно від цього фільтрувати набори даних, так щоб виводити лише необхідні дані? Як реалізувати пошук в наборі даних за певними критеріями? Як показати користувачу лише необхідні дані, замість повного набору даних? (Навести приклади з власного коду)

Як можна надати користувачу можливість вибирати певні елементи даних і залежно від цього фільтрувати набори даних, так щоб виводити лише необхідні дані?

Надати користувачу можливість вибору та фільтрації даних можна за допомогою інтерактивних елементів інтерфейсу, які дозволяють задавати критерії вибору або пошуку. Такими елементами можуть бути випадаючі списки або чекбокси комбінація з яких надає можливість отримувати відсортовані набори даних за обраними критеріями. Також можна використовувати інтуївно зрзомулі кнопки, які відразу сортуватимуть дані за певним параметром, який присутній для всіх об’єктів. Хорошим рішенням буде надати доступ користувачу шукати дані за ключовими словами.

Як реалізувати пошук в наборі даних за певними критеріями?

Для реалізації пошуку в наборі даних за певними критеріями зазвичай використовують наступний алгоритм:

1. **Отримання вхідних критеріїв пошуку**
2. **Обробка критеріїв пошуку**
   Наприклад: пошук за текстовим збігом, фільтрування за категорією, комбіновані критерії
3. **Фільтрація даних**
   Для фільтрації даних можна використовувати вбудовані функції або бази даних, наприклад SQL
4. **Виведення результатів**

Як показати користувачу лише необхідні дані, замість повного набору даних?

Щоб показати лише необхідні дані потрібно зробити зручний інтерфейс, який би їх відображав. Це можуть бути прості сторінки для яких буде використовуватись **пагінація для вже відсортованих об’єктів**.
Також можна генерувати таблицю у якій були б лише відсортовані дані.

11. Що таке віртуальні (virtual) елементи класу? Які елементи класу можуть бути віртуальними? Як віртуальні елементи пов'язані з інкапсуляцією, успадкуванням, поліморфізмом? (Навести приклади з власного коду)

Віртуальні (virtual) елементи класу — це функції або методи, які оголошуються в базовому класі з використанням ключового слова virtual та перевизначені похідним класом. Коли ви посилаєтеся на об’єкт похідного класу за допомогою покажчика або посилання на базовий клас, ви можете викликати віртуальну функцію для цього об’єкта та виконати версію методу похідного класу Віртуальними елементами класу можуть бути функції (методи), конструктори та оператори.

Віртуальні методи застосовуються для досягнення трьох основних концепцій ООП: інкапсуляції, успадкування та поліморфізму.

- Віртуальні методи дозволяють приховувати реалізацію функцій у батьківському класі від користувача. (інкапсуляція)
- Віртуальні методи дають можливість у дочірньому класі перевизначити віртуальні методи батьківського класу. Це дозволя створювати спеціалізовану логіку для конкретного похідного класу. (успадкування)
- Віртуальні методи забезпечують **динамічний поліморфізм** шляхом використання механізму **віртуальної таблиці** (*vtable*), яка дозволяє вибирати реалізацію методу в дочірньому класі під час виконання програми.

Для цього нам потрібно успадкувати sub class (дочірній клас) від base class (базовий клас) -> перевизначити віртуальні методи -> виклик методу через базовий клас

12. Як можна надати користувачу можливість вибирати певні елементи даних і залежно від цього показувати різну інформацію, яка стосується обраного елемента даних? (Навести приклади з власного коду)
Писав з голови, хуєта питання

Розглянемо декілька варіантів:
1. Консольна программа
В такому випадку можна використовувати інтерактивне текстове меню з вибором функціоналу в залежності від обраної цифри, яка обравляєтья операторами switch або if-else.
Приклад: Вибір між інтерактивним режимом, демонстраційним режом або бенчмарком в лабораторних роботах першого курсу.
2. Графічний інтерфейс
Тут вибір ширше. Ми можемо використовувати кнопки та списки, які відповідатимуть за певний функціонал та матимуть інтуїтивно зрозумілий характер. Натискання кнопки, коли ми оберемо елементи в списку, викликає відповідну функцію, яка оновлює або надає текст інформації. Також можна використати textBox для реалізації пошуку, який буде надавати відповідну інформацію, стосовно даних які ввів користувач.
3. VUI (Voice User Interface)
Користувач використовує голосові команди для вибору режимів або виконання функцій, опісля чого програма обробляє запис та надає відповідь користувачу, стосовно його запиту

13. Що таке абстрактні (abstract) класи? Що таке чисто віртуальні (pure virtual) методи? Як описати абстрактні класи та чисто віртуальні методи в мові програмування? Коли їх варто використовувати? В чому відмінність між абстрактним класом та інтерфейсом (interface)? (Навести приклади з власного коду)

Абстрактний клас — це клас, призначений спеціально для використання в якості базового класу. Абстрактний клас містить принаймні одну чисту віртуальну функцію.
Чиста **віртуальна функція** (або абстрактна функція) у C++ — це віртуальна функція, для якої ми можемо мати реалізацію, але ми повинні перевизначити цю функцію в похідному класі, інакше похідний клас також стане абстрактним класом. Чиста віртуальна функція оголошується шляхом присвоєння 0 в декларації.

```
class AB {
  public:
    virtual void f() = 0;
```

```
};
```
Функція AB::f— це чиста віртуальна функція. Оголошення функції не може мати як чистий специфікатор, так і визначення.
Абстрактні класи використовуються:
1. Створення базової моделі або шаблону
2. Гарантії реалізації методів у похідних класах
3. Коли не потрібна повна реалізація базового класу
4. Поліморфізм
Різниця між абстрактним класами та інтерфейсами в C#:

	Abstract Class	Interface
Успадкування	Клас може успадковувати тільки один абстрактний клас	Клас може реалізовувати багато інтерфейсів, що дозволяє досягати множинного успадкування.
Конструктори	Може мати конструктори.	Не може мати конструкторів.
Модифікатори доступу	Може використовувати різні модифікатори доступу (public, protected, internal, private).	За замовчуванням усі члени є public. Починаючи з C# 8.0, інтерфейси можуть містити приватні методи, але інші модифікатори доступу
Методи з реалізацією	Може містити як абстрактні методи (без реалізації), так і методи з реалізацією.	Може містити оголошення методів без реалізації та методи з реалізацією за замовчуванням
Поля	Може мати поля (змінні) будь-якого	Не може мати полів (змінних), але може

	типу.	мати властивості (properties).

14. Якщо в програмі зберігається забагато інформації, і одразу показувати користувачу всю інформацію недоцільно – як організувати інтерфейс користувача, щоб зручним чином надати доступ до всієї інформації і при цьому не перевантажувати користувача даними? (Навести приклади з власного коду). (GPT, але норм)

В такому разі є декілька варіантів вирішення проблеми:
1. Використання вкладок (tabs)
Потрібно розбити інформацію на логічні секції і розмістіть її у вкладках. Це дозволяє користувачу перемикатися між різними категоріями, не бачачи одразу всю інформацію.
Приклад: Вкладки для "Профіль", "Налаштування", "Статистика".

2. Панелі навігації або меню
Зробіть багаторівневу структуру, використовуючи бічне меню або випадаючі списки для доступу до різних розділів інформації.
Приклад: Головна сторінка з коротким оглядом і бокове меню для доступу до глибших рівнів.

3. Пагінація (pagination)
Розбийте інформацію на сторінки з можливістю перегортання.
Приклад: Таблиця з даними, де показується по 10 записів на сторінку.

4. Фільтри та пошук
Додайте можливість пошуку чи фільтрації інформації за ключовими словами або критеріями.
Приклад: Введення ключового слова для пошуку конкретних документів у великій базі даних.

5. Підказки та контекстна допомога
Додайте підказки (tooltip) або кнопки "?" для доступу до пояснень чи додаткової інформації.

Приклад: Наведення на значок питання поруч з полем показує контекстну інформацію.

6. Пріоритезація (focus on relevance)
Показуйте найбільш важливу інформацію першочергово, а другорядні дані приховуйте або робіть доступними через додаткові дії (наведення, кнопка
Приклад: На екрані замовлення спочатку показується статус, а потім приховані деталі.

15. Що таке конструктор? Для чого використовуються конструктори? Чи всі класи мають містити конструктори? Чи може клас містити декілька конструкторів? Чи можуть конструктори бути віртуальними, і якщо так - коли це використовується? Коли і в якому порядку викликаються конструктори? (Навести приклади з власного коду).

Що таке конструктор?
Конструктор у C++ — це спеціальний метод, який автоматично викликається під час створення об’єкта класу.
Для чого використовуються конструктори? Чи всі класи мають містити конструктори?
Він зазвичай використовується для ініціалізації членів даних нових об’єктів. Конструктор у C++ має те саме ім’я, що й клас або структура. Він створює значення, тобто надає дані для об’єкта, тому його називають конструктором.
Чи може клас містити декілька конструкторів?
Усі класи в C++ можуть мати декілька конструкторів, але це не обов'язково. Якщо клас не має явно визначеного конструктора, компілятор автоматично створює конструктор за замовчуванням, який не виконує жодних дій (просто викликає конструктори за замовчуванням для членів класу). Однак, ми можемо створити власні конструктори, які будуть створювати об’єкт за певним “правилом”. Тоді компілятор більше не створює автоматичного конструктора за замовчуванням.

Чи можуть конструктори бути віртуальними, і якщо так - коли це використовується?
У C++ ми **НЕ МОЖЕМ** зробити конструктор віртуальним. Причина полягає в тому, що C++ є статично типізованою мовою, тому концепція

віртуального конструктора є суперечливою, оскільки компілятор повинен знати точний тип під час компіляції, щоб виділити правильний обсяг пам’яті та правильно ініціалізувати об’єкт.
Якщо ми спробуємо зробити конструктор віртуальним, компілятор позначить помилку.

Коли і в якому порядку викликаються конструктори?

Конструктори викликаються при створенні нового об'єкта у сегменті даних, стеку, динамічній пам'яті та ініціалізації його полів.
Якщо два класи утворюють ієрархію спадковості, то при створенні екземпляру похідного класу спочатку викликається конструктор базового класу який конструює об’єкт похідного класу. Потім цей конструктор стає недоступним і доповнюється кодом конструктора похідного класу. Таким чином, спочатку відбувається ініціалізація даних базового класу, потім відбувається ініціалізація даних похідного класу.

16. Що таке деструктор? Для чого використовуються деструктори? Чи всі класи мають містити деструктори? Чи може клас містити декілька деструкторів? Чи можуть деструктори бути віртуальними, і якщо так - коли це використовується? Коли і в якому порядку викликаються деструктори? (Навести приклади з власного коду)

Деструктор — це функція-член екземпляра, яка автоматично викликається щоразу, коли об’єкт буде знищено. Це означає, що деструктор є останньою функцією, яка буде викликана перед знищенням об’єкта.
(!) Його основна мета — виконати очищення ресурсів, які були виділені під час життя об'єкта, наприклад, закрити файли, звільнити пам'ять або зняти блокування.
Якщо ми не пишемо власний деструктор у класі, компілятор створює для нас деструктор за замовчуванням. **Деструктор за замовчуванням працює добре, якщо ми не маємо динамічно виділеної пам’яті або покажчика в класі.** Коли клас містить вказівник на пам’ять, виділену в класі, ми повинні написати деструктор, щоб звільнити пам’ять до того, як екземпляр класу буде знищено. Це необхідно зробити, щоб уникнути витоків пам'яті.
- Чи може клас містити декілька деструкторів?
Ні, у класі може бути лише один деструктор із назвою класу, якому передує ~, без параметрів і без типу повернення.
- Чи можуть деструктори бути віртуальними, і якщо так - коли це використовується?
Так. Насправді, якщо у нас є віртуальна функція, це завжди гарна ідея зробити деструктори віртуальними в базовому класі.
(!) Якщо деструктор у базовому класі **не є віртуальним**, то під час знищення об'єкта через вказівник на базовий клас буде викликаний лише деструктор базового класу, що призведе до витоків ресурсів у похідному класі.
Коли і в якому порядку викликаються деструктори?
Коли об'єкт виходить за межі області або видаляється, послідовність подій при його повному знищенні виглядає наступним чином:

1. Викликається деструктор класу, і виконується тіло функції деструктора.
2. Деструктори для об'єктів нестатичних членів викликаються у порядку, зворотному до порядку їх появи в оголошенні класу. Необов'язковий список ініціалізації елементів, що використовується

під час створення цих елементів, не впливає на порядок їх створення або знищення.

3. Деструктори для невіртуальних базових класів викликаються у зворотному порядку оголошення.
4. Деструктори для віртуальних базових класів викликаються у порядку, зворотному до порядку їх оголошення.

17. Що таке модифікатор const? Де він використовується і що означає? Коли доцільно його використовувати? Чи зміниться поведінка програми, якщо його прибрати? (Навести приклади з власного коду)

У деяких мовах програмування (зокрема C, C++) const є ключовим словом, застосованим до типу даних, яке вказує, що дані доступні лише для читання, тобто значення змінної, об'єкта або параметра не може бути змінено після ініціалізації.

(Хоча це можна використовувати для оголошення констант, const у сімействі мов C відрізняється від подібних конструкцій в інших мовах тим, що є частиною типу, і тому має складну поведінку в поєднанні з покажчиками, посиланнями, складеними даними типи та перевірка типів.)

В інших мовах дані не зберігаються в одному місці пам’яті, а копіюються під час компіляції для кожного використання.

18. Що таке керування пам'яттю (memory management)? Що таке втрати пам'яті (memory leaks)? Як правильно створювати та знищувати об'єкти, щоб не виникало проблем? (Навести приклади з власного коду)

Управління пам’яттю (memory management) є критично важливим аспектом мов програмування, який передбачає виділення та звільнення пам’яті під час виконання програми. Тобто, це процес керування тим, скільки пам’яті використовує програма для виконання різних операцій.

Витоки пам’яті (memory leaks) відбуваються, коли виділена пам’ять у програмі не розподіляється належним чином, що призводить до втрати ресурсів пам’яті. Іншими словами, витік пам’яті відбувається, коли програма не може звільнити пам’ять, яка більше не потрібна, що призводить до накопичення невикористаної пам’яті з часом. Це може мати

шкідливий вплив на продуктивність програмного забезпечення та загальну стабільність системи.

Витоки пам'яті особливо неприємні, оскільки вони можуть залишатися непоміченими протягом тривалого часу, поступово знижуючи продуктивність програмної системи. Оскільки витоки пам’яті продовжуються, доступна пам’ять вичерпується, що призводить до збоїв і уповільнення роботи системи.

Поширені причини меморі лікс: (не обов’язково писати)

1. Однією з поширених причин витоків пам’яті є неправильне керування пам’яттю. Це може статися, коли програміст забуває звільнити виділену пам’ять або не справляється належним чином із умовами помилки, які можуть перешкодити звільненню пам’яті. У таких випадках пам’ять залишається виділеною, навіть якщо вона більше не потрібна.

2. Іншою причиною витоку пам'яті є неправильне використання динамічного розподілу пам'яті. Динамічний розподіл пам'яті дозволяє програмам запитувати пам'ять під час виконання, але якщо це не використовується належним чином, це може призвести до витоку пам'яті. Наприклад, якщо програма виділяє пам’ять, але не звільняє її після використання, може статися витік пам’яті.

3. Крім того, витоки пам’яті можуть бути викликані циклічними посиланнями або циклічними залежностями. Це відбувається, коли об’єкти посилаються один на одного таким чином, що запобігає звільненню їх пам’яті. Якщо не керувати належним чином, ці циклічні посилання можуть призвести до витоків пам’яті та погіршити продуктивність програмної системи.

(GPT):
Щоб уникнути проблем із пам'яттю, таких як витоки, важливо правильно створювати та знищувати об'єкти. Ось кілька ключових підходів і принципів:

1. Дотримання принципу "створив – знищив".

Для кожного виділення пам’яті (наприклад, через `new` у C++ або `malloc` у C) має бути відповідна операція звільнення (`delete` або `free`).

2. Автоматизація через збирач сміття (Garbage Collection).

У мовах із вбудованим garbage collector пам'ять для об’єктів звільняється автоматично.

3. Виявлення витоків на ранніх етапах.

Використання інструментів для аналізу пам’яті.

Варто запускати тестування, орієнтоване на виявлення проблем із пам'яттю.

4. Робота з динамічною пам’яттю.

Мінімізуйте використання ручного керування пам’яттю. Натомість використовуйте у C++: `std::unique_ptr` або `std::shared_ptr` для безпечного керування динамічними ресурсами.

Завжди перевіряйте результати виділення пам’яті (наприклад, чи не повернуло `NULL`).

5. Уникнення циклічних посилань.

У структурах даних, що можуть містити цикли (наприклад, графи), використовуйте слабкі посилання (`weak_ptr` у C++).

6. Обробка виключень.

Переконайтеся, що пам’ять звільняється навіть у разі помилок або виключень (у C++ це реалізується через деструктори).

7. Використання сучасних підходів до проєктування.

У сучасному програмуванні рекомендується уникати прямого керування пам’яттю там, де це можливо:

Віддавайте перевагу контейнерам стандартних бібліотек (наприклад, `std::vector` у C++).

Використовуйте фреймворки, які забезпечують автоматизацію керування ресурсами.

Дотримання цих принципів допоможе уникнути проблем із пам'яттю та забезпечить стабільну та ефективну роботу програм.

19. Що таке "розумні вказівники" (smart pointers)? Для чого їх використовувати? Які переваги та недоліки їх використання, порівняно зі звичайними вказівниками (raw pointers), посиланнями (references) та просто екземплярами об'єктів? (Навести приклади з власного коду).
19.1 Що таке "розумні вказівники" (smart pointers)? Які бувають різновиди розумних вказівників?....(все те саме)

Розумний вказівник — це клас-обгортка над вказівником з перевантаженими операторами, такими як * і ->. Об'єкти класу розумного вказівника виглядають як звичайні вказівники. Але, на відміну від

звичайних вказівників, вони можуть автоматично звільняти пам'ять для знищеного об'єкта.
Різновиди розумних покажчиків:
- ***auto_ptr*** дозволяє керувати об’єктами, отриманими за допомогою оператора new, і автоматично видаляє їх, коли сам auto_ptr знищується. Коли об’єкт описується за допомогою auto_ptr, він зберігає вказівник на один виділений об’єкт.
Обмеження: auto_ptr передає право власності при копіюванні, що може спричинити небажані побічні ефекти. Через це він вважається застарілим і не рекомендується до використання.
- *unique_ptr* зберігає тільки один вказівник. Це ексклюзивний власник ресурсу: якщо потрібно прив’язати новий об’єкт, спочатку слід звільнити поточний.(іншими словами якщо треба змінити власника даних то це можна реалізувати через додаткові функції такі як std::move(), srd::swap())
Особливості:
1) Унікальний контроль над ресурсом.
2) Неможливо скопіювати, але можна переміщувати (std::move).
3) Легке та ефективне управління ресурсами.|
- ***shared_ptr*** дозволяє декільком вказівникам одночасно посилатися на один і той самий об’єкт. Він використовує лічильник посилань, щоб відстежувати кількість вказівників, які посилаються на об’єкт.
Особливості:
1) Лічильник посилань можна отримати за допомогою методу use_count().
2) Об’єкт автоматично знищується, коли лічильник досягає нуля.
3) Зручний для спільного доступу до одного ресурсу.

//Цей вказівник на лекції не давав але написати мона

- weak_ptr — це розумний вказівник, який утримує **неволодіюче** посилання на об’єкт. Він схожий на shared_ptr, але не збільшує лічильник посилань.
Призначення:
1) Використовується для уникнення **циклічних залежностей**, які можуть виникнути, якщо два чи більше об'єкти посилаються один на одного через shared_ptr.
2) Наприклад, у ситуаціях із батьківсько-дочірніми залежностями, коли об’єкти посилаються один на одного.
Особливості:

1) Не збільшує лічильник посилань.
2) Не гарантує, що об’єкт, на який посилається, все ще існує — необхідно перевіряти це за допомогою методу expired() перед доступом.
3) Для отримання контрольованого ресурсу використовується метод lock().

Для чого їх використовувати?
Розумні вказівники використовуються для автоматизованого та безпечного управління ресурсами, такими як пам’ять або системні дескриптори, у програмах на C++. Вони дозволяють уникати витоків пам’яті та спрощують управління життєвим циклом об’єктів. Основна причина їх використання — це імплементація принципу **RAII**

Які переваги та недоліки їх використання, порівняно зі звичайними вказівниками (raw pointers), посиланнями (references) та просто екземплярами об'єктів?
Переваги розумних вказівників: Автоматичне управління пам’яттю, Безпечне управління життєвим циклом об’єктом, RAII (Resource Acquisition Is Initialization): **автоматичне очищення:** за допомогою RAII
Недоліки розумних вказівників: Використання додаткових ресурсів