Тема
Наслідування. Створення та використання ієрархії класів.

Мета
Навчитися створювати базові та похідні класи, використовувати наслідування різного типу доступу, опанувати принципи використання множинного наслідування. Навчитися перевизначати методи в похідному класі, освоїти принципи такого перевизначення.

Теоретичні відомості
Наслідування
Наслідуванням називається процес визначення класу на основі іншого класу. На новий (дочірній) клас за замовчуванням поширюються всі визначення змінних екземпляра і методів зі старого (батьківського) класу, але можуть бути також визначені нові компоненти або «перевизначені» визначення батьківських функцій і дано нові визначення. Прийнято вважати, що клас А успадковує свої визначення від класу В, якщо клас А визначений на основі класу В зазначеним способом.

Класи можуть бути пов'язані один з одним різними відношеннями. Одним з основних є відношення клас-підклас, відоме в об'єктно-орієнтованому програмуванні як наслідування. Наприклад, клас автомобілів Audi 6 є підкласом легкових автомобілів, який в свою чергу входить у більший клас автомобілів, а останній є підкласом класу транспортних засобів, який крім автомобілів включає в себе літаки, кораблі, потяги і т.д. Прикладом подібних відношень є системи класифікації в ботаніці та зоології.

При наслідуванні всі атрибути і методи батьківського класу успадковуються класом-нащадком. Наслідування може бути багаторівневим, і тоді класи, що знаходяться на нижніх рівнях ієрархії, успадкують всі властивості (атрибути і методи) всіх класів, прямими або непрямими нащадками яких вони є.

Крім одиничного, існує і множинне наслідування, коли клас наслідує відразу кілька класів. При цьому він успадкує властивості всіх класів, нащадком яких він є.

При наслідуванні одні методи класу можуть заміщатися іншими. Так, клас транспортних засобів буде мати узагальнений метод руху. У класах-нащадках цей метод буде конкретизований: автомобіль буде їздити, літак – літати, корабель – плавати. Така зміна семантики методу називається поліморфізмом. Поліморфізм – це виконання методом з одним і тим же ім'ям різних дій залежно від контексту, зокрема, від приналежності до того чи іншого класу. У різних мовах програмування поліморфізм реалізується різними способами.

Синтаксис наслідування класів:

```
class Animal
{
public:
    unsigned int GetAge() const { return m_uiAge; }
    void SetAge(unsigned int uiAge) { m_uiAge = uiAge; }

private:
    unsigned int m_uiAge{0};
};

class Horse : public Animal
{};
```

Рис. 1. Приклад наслідування класів

Private члени класу недоступні для наслідування. Звісно можна було б їх зробити public, та в такому випадку вони будуть доступні всім класам які знаходяться в програмі. Для того, щоб доступ до членів класу мали тільки класи нащадки потрібно використовувати модифікатор доступу protected.

Дивлячись на приклад коду вище – правдиве твердження – об’єкти класу Horse є також і об’єктами класу Animal. В цьому полягає суть ієрархічних відношень між класами. Коли в класі Horse створюється об’єкт, то для цього з класу Animal викликається базовий конструктор і він викликається першим. Після цього викликається конструктор класу Horse, який завершує створення об’єкта. Об’єкт класу Horse не буде існувати поки повністю не буде завершене його створення обома конструкторами.

При видаленні об’єкта класу Horse з пам’яті комп’ютера спочатку викликається деструктор класу Horse а потім деструктор класу Animal.

Розглянемо наступний код:

```
class Mammal
{
public:
    Mammal(): itsAge(0) {}
    explicit Mammal(unsigned int age): itsAge(age) {}

protected:
    unsigned int itsAge;
};

enum class BREED
{
    Unknown,
    FrenchBulldog,
    GoldenRetriever,
    GermanShepherd,
    Poodle
};

class Dog: public Mammal
{
public:
    Dog(): itsBreed(BREED::Unknown) {}
    explicit Dog(BREED breed): itsBreed(breed) {}

protected:
    BREED itsBreed;
};
```

Рис. 2. Використання конструкторів при наслідуванні.

Ми перевантажили конструктори за замовчуванням в класах Mammal та Dog, таким чином, що перший з них присвоює об’єкту вік а другий породу. Як нам створити об’єкт класу Dog так, щоб ми могли використати конструктор базового класу, передати йому значення віку і проініціалізувати його? Для цього створимо конструктор в класі Dog який буде використовувати базовий конструктор:

```
Dog(int age, BREED breed): Mammal(age), itsBreed(breed) {}
```

Рис. 3. Ініціалізація базового класу

Заміщення функцій

Об’єкт класу Dog має доступ до всіх функцій-членів класу Mammal а також до будь-якої функції члену яка оголошена в цьому ж класі. Крім цього базові функції можуть бути заміщені в похідному класі. Під заміщенням базової функції розуміють зміну її виконання в похідному класі.

Для заміщення необхідно описати функцію в похідному класі з таким же ж іменем як у базовому.

```cpp
#include <iostream>

class Mammal
{
protected:
    int itsAge{0};

    void Speak() { std::cout << «speak»; }
};

class Dog : public Mammal
{
public:
    void Speak() { std::cout << «Wof!!!»; }
};

int main(int, char **)
{
    Dog dog;
    dog.Speak();

    return 0;
}
```

Рис. 4. Заміщення функцій у похідному класі

В такому випадку нам на екран виведеться Wof!!!.

Недоліки заміщення

Якщо в базовому класі у нас є перевантажена функція, яку ми хочемо замінити в похідному класі – ми не зможемо викликати в похідному класі будь-яку з цих перевантажених функцій:

```cpp
#include <iostream>

class Mammal
{
protected:
    unsigned int itsAge{0};
    void Speak() const { std::cout << "speak"; }
    void Speak(int number) const { std::cout << "speak" << number; }
};

class Dog : public Mammal
{
public:
    void Speak() const { std::cout << "Wof!!!"; }
};

int main(int, char **)
{
    Dog dog;
    dog.Speak();
    dog.Speak(1);

    return 0;
}
```

Рис. 5. Приховування функцій базового класу при заміщенні функції у похідному

В підкресленому рядку буде помилка при компіляції.

Для того, щоб викликати даний метод нам потрібно також замінити цей варіант перевантаження в похідному класі та в його реалізації викликами метод базового класу. Також можна виконати додаткові дії, специфічні для похідного класу:

```cpp
#include <iostream>

class Mammal
{
protected:
    unsigned int itsAge{0};
    void Speak() const { std::cout << "speak"; }
    void Speak(int number) const { std::cout << "speak" << number; }
};

class Dog : public Mammal
{
public:
    void Speak() const { std::cout << "Wof!!!"; }
    void Speak(int number) const { Mammal::Speak(number); std::cout << "child method"; }
};

int main(int, char**)
{
    Dog dog;
    dog.Speak();
    dog.Speak(1);
    return 0;
}
```

Рис. 6. Виклик прихованої функції базового класу через її заміщення у похідному

Тут в заміщенні функції ми викликаємо базовий варіант цієї функції і розширюємо його додатковим функціоналом.

Або ж ми можемо викликати цей метод із базового класу. Для цього необхідно вказати область видимості базового класу.

```cpp
#include <iostream>

class Mammal
{
protected:
    unsigned int itsAge{0};
    void Speak() const { std::cout << "speak"; }

public:
    void Speak(int number) const { std::cout << "speak" << number; }
};

class Dog : public Mammal
{
public:
    void Speak() const { std::cout << "Wof!!!"; }
};

int main(int, char**)
{
    Dog dog;
    dog.Speak();
    dog.Mammal::Speak(1);
}
```

Рис. 7. Виклад прихованої функції базового класу через область видимості

В даному випадку метод з базового класу потрібно зробити public.

Множинне наслідування

В мові програмування c++ є можливість множинного наслідування. Приклад використання такого наслідування:

```cpp
#include <iostream>

class Animal
{
private:
    unsigned int age{0};

public:
    unsigned int GetAge() const { return age; }
    void SetAge(unsigned int value) { age = value; }
};

class Horse : public Animal
{
public:
    void Gallop() const { std::cout << "Gallop !!!"; }
};

class Bird : public Animal
{
public:
    void Fly() const { std::cout << "Fly"; }
};

class Pegasus : public Horse, public Bird
{
public:
    void Chirp() const { std::cout << "Whinny!"; }
};
```

Рис. 8. Приклад використання множинного наслідування

При створення об’єкта класу Pegasus будуть викликатись конструктори класів Horse і Bird саме в такій послідовності. Вкінці викличеться конструктор класу Pegasus.

Параметр рівня доступу при наслідуванні

При наслідуванні члени базового класу стають членами похідного класу. Як правило, для наслідування використовується наступна синтаксична конструкція.

```cpp
class ім'я_похідного_класу : рівень_доступу ім'я_базового_класу
{
    // тіло класу
}
```

Рис. 9. Синтаксична конструкція наслідування із вказанням рівня доступу

Рівень доступу визначає статус членів базового класу в похідному класі. Як цей параметр використовуються специфікатори public, private або protected. Якщо рівень доступу не вказаний, то для похідного класу за умовчанням використовується специфікатор private, а для похідної структури - public.

Розглянемо варіанти, що виникають в цих ситуаціях. Якщо рівень доступу до членів базового класу задається специфікатором public то всі відкриті і захищені члени базового класу стають відкритими і захищеними членами похідного класу. При цьому закриті члени базового класу не міняють свого статусу і залишаються недоступними членам похідного.

Якщо властивості базового класу успадковуються за допомогою специфікатора доступу private, всі відкриті і захищені члени базового класу стають закритими членами похідного класу. При закритому наслідуванні всі відкриті і захищені члени базового класу стають закритими членами похідного класу. Це означає, що вони залишаються доступними членам похідного класу, але недоступні решті елементів програми, що не є членами базового або похідного класів.

Специфікатор protected підвищує гнучкість механізму наслідування. Якщо член класу оголошений захищеним (protected), то поза класом він недоступний. З цієї точки зору захищений член класу нічим не відрізняється від закритого. Єдине виключення з цього правила стосується наслідування. У цій ситуації захищений член класу істотно відрізняється від закритого. Як вказувалося вище, закритий член базового класу не доступний іншим елементам програми, включаючи похідний клас. Проте захищені члени базового класу поводяться інакше. При відкритому наслідуванні захищені члени базового класу стають захищеними членами похідного класу до отже, доступні решті членів похідного класу. Іншими словами захищені члени класу по відношенню до свого класу є закритими і в той же час, можуть успадковуватися похідним класом.

Завдання

1. Розробити ієрархію класів відповідно до варіанту. Набір полів і методів, необхідних для забезпечення функціональної зручності класів, визначити самостійно.
2. Створити базовий (відповідає сутності) та похідні класи (типи сутностей).
3. Використати public, protected наслідування.
4. Використати множинне наслідування (за необхідності).
5. Визначити функцію PrintName() в базовому класі, яка друкує назву відповідного класу. Перевизначити її в похідних класах.
6. Реалізувати методи варіанта та результати вивести у файл.
7. Продемострувати роботу класів таким чином (на вибір):
    a. За допомогою тестів Google Test, які перевірятимуть роботу ієрархії класів.
    b. За допомогою інтерактивної програми із демострацією її роботи.
8. Оформити звіт до лабораторної роботи. Включити у звіт UML-діаграму розробленої ієрархії класів та результат роботи тестів.

Варіанти

Варіант	Завдання
1	Розробити ієрархію класів для сутності: **кредит** (CLoan)**.** Розробити такі типи кредитів: - Кредит, при якому сума ділиться рівними платежами (CEqualPaymentLoan); - Кредит, при якому нараховується відсоток від суми залишку (CPercentPaymentLoan); - Пільговий кредит, при якому держава компенсує частину відсотків по кредиту (CConcessionalLoan). Класи повинні мати повний набір методів для роботи з ними. Кожен клас обов'язково повинен вміти обчислити суму платежу в заданий місяць, суму виплачену до заданого місяця, суму, яка буде виплачена за весь період.
2	Розробити ієрархію класів для сутності: **банківський рахунок** (CBankAccount). Розробити наступні типи банківських рахунків: - Звичайний – стандартна комісія на оплату комунальних послуг, перерахунок на інший рахунок, зняття готівки (CStandardBankAccount). - Соціальний – оплата комунальних послуг безкоштовна, відсутня комісія за зняття готівки (пенсії), нараховується невеликий відсоток з залишку на картці. (CSocialBankAccount). - Преміум – наявність кредитного ліміту, низький відсоток за користування кредитним лімітом, нараховується більший відсоток, якщо залишок на картці більший за якусь суму. (CPremiumBankAccount). Кожен із рахунків повинен зберігати історію транзакцій.
3	Розробити ієрархію класів для сутності: **банківський депозит** (CDeposit). Розробити такі типи депозитів:

	- Строковий – виплата відсотків відбувається після закінчення терміну депозиту (CFixedDeposit); - Накопичувальний – капіталізація відсотків, виплата відбувається кожного місяця (CAccumulativeDeposit); - VIP – капіталізація відсотків, виплата кожного місяця, можливість поповнення рахунку в будь-який день, збільшення відсоткової ставки із заданим коефіцієнтом при збільшенні суми вкладу (обмежене зверху) (CVIPDeposit). Всі класи повинні вміти обчислювати прибуток за вказаний та за весь період вкладу.
4	Розробити ієрархію класів для сутності: **облік спожитої електроенергії** (CElectricityConsumption). Розробити такі моделі обліку: - Звичайну – вартість спожитої електроенергії обчислюється за фіксованою ціною (CFixedElectricityConsumption); - Пільгову – вартість спожитої електроенергії обчислюється за пільговою ціною у випадку, коли обсяг не перевищує нормованого (CSocialElectricityConsumption). - Багатозонну – вартість спожитої електроенергії обчислюється за підвищеною ціною в час-пік, за зниженою ціною в нічний час і за стандартною в іншому випадку (CMultiZoneElectricityConsumption) Кожен клас повинен мати можливість обліковувати спожиту електроенергію погодинно, обчислювати вартість спожитої електроенергії за заданий період, а також зберігати та завантажувати журнал обліку з файлу.
5	Розробити ієрархію класів для сутності: **геометрична фігура** (CShape). Розробити такі похідні класи: - Прямокутник (CRectangle) - Круг (CCircle) - Трикутник (CTriangle) Кожен клас повинен мати можливість обчислювати свою площу та периметр.
6	Розробити ієрархію класів для сутності: **геометричне тіло** (CSolidFigure). Розробити такі похідні класи: - Циліндр (CCylinder) - Сфера (CSphere) - Куб (CCube) Кожен клас повинен мати можливість обчислювати свою площу та об’єм.
7	Розробити ієрархію класів для сутності: **тварина** (CAnimal). Розробити такі похідні класи: - Кіт (CCat) - Пес (CDog) - Кінь (CHorse) Кожен клас повинен мати можливість обчислювати калорійність продуктів харчування необхідних на один день життя.
8	Розробити ієрархію класів для сутності: **оплата** (CPayment) - Оплата кредитною картою (CCreditCardPayment) - PayPal-сервісна оплата (CPayPalPayment) - Готівкова оплата (CCashPayment) Кожен клас повинен мати можливість задавати процес оплати (задання реквізитів, суми, обчислення комісії, обмеження на максимальну суму переказу)
9	Розробити ієрархію класів для сутності: **лікар** (CDoctor) - Дитячий лікар (CPediatrician) - Терапевт (CTherapist)

	- Стоматолог (CDentist) Кожен клас має мати протокол ведення огляду. Наприклад, дитячий лікар повинен виявити щеплення, перевірити стан розвитку і т.ін., стоматолог – перевірити стан ротової порожнини, залежно від результату огляду – зробити рентген, терапевт – виміряти тиск, поміряти пульс, і.т.ін. Залежно від результату – призначити збір аналізу, чи огляд спеціаліста вузького профілю.
10	Розробити ієрархію класів для сутності: **поштове відправлення** (CParcel). Розробити наступні типи відправлень - Бандероль звичайна (CCommonParcel); - Бандероль із оголошеною цінністю (CDeclaredValueParcel); - Електронний переказ (CDigitalTransfer). Класи повинні мати повний набір методів для роботи з ними. Кожен клас обов'язково повинен вміти обчислити вартість відправлення (як приклад тарифів можна взяти реальні тарифи тут: http://ukrposhta.ua/dovidka/tarifi/inshi-poslugi).
11	Розробити ієрархію класів для сутності: **працівник** (CEmployee). Розробити такі типи працівників: - Менеджер (CManagerEmployee) - Продавець (CSalesmanEmployee) - Інженер (CEngineerEmployee) Кожен клас обов'язково повинен вміти обчислити заробітну плату для себе. Заробітна плата менеджера має залежати від кількості підлеглих працівників. Заробітна плата продавця залежить від кількості продуктів, які він продає.
12	Розробити ієрархію класів для сутності: **їжа** (CFood). Розробити такі типи їжі: - Куряче м'ясо (CChickenMeat) - Молоко (CMilk) - Куряче яйце (CChickenEgg) Кожен клас обов'язково повинен вміти обчислити калорійність на 1 кг продукту. Також кожен клас повинен вміти обчислити максимальну рекомендовану масу споживання за день для особи із заданою масою (щоб не викликати негативних наслідків для здоров'я).
13	Розробити ієрархію класів для сутності: **транспортний засіб** (CVehicle). Розробити такі типи транспортних засобів: - автомобіль (CCar) - вантажівка (CTruck) - мотоцикл (CMotorcycle) Кожен клас повинен вміти розраховувати: - відстань (км), яку може проїхати за 1 літр палива із корисним навантаженням (кг); - витрати на обслуговування (за 1000 км). - максимальну корисну масу перевезення.
14	Розробити ієрархію класів для сутності: **шахова фігура** (CChessman) Розробити такі типи сортувальників: - Кінь (CKnight) - Ферзь (CQueen) - Тура (CRook) Кожен клас повинен мати метод, який перевіряє, чи б'є фігура задану позицію.
15	Розробити ієрархію класів для сутності: **геометричне тіло** (CSolidFigure). Розробити такі типи фігур: - пряма призма (CRightPrism);

	- паралелепіпед (CParallelepiped); - правильний тетраедр (CRegularTetrahedron). Кожен клас повинен вміти обчислювати об’єм та площу поверхні фігури.