Лабораторна робота № 6. Одинарне успадкування

**Мета роботи**: вивчення способів та механізмів одинарного успадкування.
Теоретичний матеріал лекції, [1, розділ 6], [2, розділ 16]; [3, розділ 13 ].

## 1. Короткі теоретичні відомості

### 1.1. Суть успадкування класів

Успадкування класів і структур є одним із способів повторного використання коду. Успадкування полягає у використанні *оголошень* елементів базового класу у структурі іншого класу, який називається похідним від базового. У результаті успадкування об'єкт похідного класу набуває властивостей базового класу.

Похідний клас може користуватися полями даних та методами, успадкованими від базового класу без їх повторного оголошення. Крім того, похідний клас може оголосити власні поля даних та методи. Оголошення елементів похідного класу перекривають однойменні оголошення елементів базового класу.

Поля даних базового класу не копіюються у похідний клас. На машинному рівні для обєкта похідного класу буде виділено окрему від базового класу область пам'яті під успадковані поля.

Конструктори, деструктори, операторна функція присвоєння та друзі класу не успадковуються.

Між обєктами представниками базового та похідного класу діє відношення "є": похідний клас є нащадком базового класу, а базовий клас є його предком. В інформаційному плані нащадок є багатшим від предка.

### 1.2. Оголошення успадкування

Для успадкування базового класу необхідно після імені базового класу через символ ':' вказати режим успадкування та ім'я базового класу:

```
class Base { . . . };
class Derived: режим_успадкування Base { . . . };
```

Режим успадкування може мати наступні значення: `private, protected, public`. Режим успадкування визначає, в які частини похідного класу потрапляють елементи базового класу.

При `private`-успадкуванні елементи всіх частин базового класу потрапляють у `private`-частину похідного класу.

При `protected`-успадкуванні `private`-частина базового класу потрапляє у `private`-частину похідного класу, а `protected`- та `public`-частини базового класу потрапляють у `protected`-частину похідного класу.

При `public`-успадкуванні `private, protected` та `public`-частини базового класу потрапляють в однойменні частини похідного класу.

Для будь-якого варіанта успадковуються всі елементи базового класу у похідний клас, але `private`-частина базового класу безпосередньо недоступна у похідному класі.

Схеми успадкування класів показано в табл. 1.

Для ініціалізації успадкованих полів викликається конструктор базового класу у списку ініціалізації конструктора похідного класу (після символу ':').

У похідному класі існує можливість зміни застосованого режиму успадкування елементів базового класу. Успадковані елементи можна перевести тільки на той самий рівень, на якому вони були розміщені в базовому класі. Для цього в області дії цього рівня у похідному класі необхідно виконати оголошення:

```
ім'я_базового_класу::імя_елемента;
```

**Таблиця 1**. **Дія режимів успадкування класів**

| Режим успадкування | Рівні захисту класу Base | Рівні захисту класу Derived |
|---|---|---|
| private | private | private* |
| | protected | private |
| | public | private |
| protected | private | private* |
| | protected | protected |
| | public | protected |
| public | private | private* |
| | protected | protected |
| | public | public |

У табл. 1.1 символ ‘*’ позначає `private`-частину базового класу, безпосередньо не доступну у похідному класі.

### 1.4. Перекриття елементів класу

Елементи похідного класу перекривають дію однойменних елементів базового класу. Якщо у будь-якій частині похідного класу містяться елементи, імена яких збігаються з успадкованими елементами базового класу, то відбувається перекриття таких успадкованих елементів. Поля даних похідного класу перекривають однойменні дані базового класу. Для доступу до перекритих полів базового класу у похідному класі використовується конструкція:
`ім’я_базового_класу::ім’я_поля;`

Методи похідного класу перекривають однойменні методи базового класу. Перекриття методів реалізується через їх перевизначення (`override`) або перевантаження (`overload`). Перевизначені функції мають однакове ім’я, однакову кількість та типи параметрів, однаковий тип результату. Перевизначення допускається між успадкованими класами, але не допускається у межах одного класу.

Для доступу до перекритих методів використовується ім’я класу, де визначений метод, та операція дозволу доступу:

```
ім’я_базового_класу::ім’я_методу(фактичні_аргументи);
```

Перевантажені функції мають однакове ім’я, можуть мати однаковий або різний тип результату, але обов’язково відрізняються кількістю або типами своїх параметрів. Перевантаження допускається у межах одного класу та між успадкованими класами. Вибір конкретного перевантаженого методу визначається списком фактичних аргументів.

### 1.5. Особливості успадкування закритої частини базового класу

Хоча `private`-частина базового класу успадковується похідним класом, але її елементи безпосередньо не доступні для використання у похідному класі. За необхідності такий доступ може бути реалізований одним із способів:

- через методи базового класу, розміщені у `protected` або `public`-частинах базового класу;
- оголошенням похідного класу другом до базового класу. Якщо клас нащадок є дружнім до базового класу, то він має доступ до елементів усіх частин класу за їх іменами, а не за допомогою об’єкта, як це забезпечується для дружніх зовнішніх класів та функцій.

### 1.6. Порядок виклику конструкторів та деструкторів при успадкуванні класів

Основна задача конструктора полягає в ініціалізації полів-даних об’єктів класу. За необхідності у конструкторі можна виділити динамічну пам’ять для даних об’єкта. Якщо клас містить оголошення віртуальних функцій, то конструктор додатково виконує ініціалізацію вказівника на таблицю віртуальних методів.

При успадкуванні діє такий порядок виклику конструкторів:

- викликається конструктор базового класу; якщо явний виклик конструктора базового класу не здійснюється у списку ініціалізації конструктора похідного класу (після двокрапки), то викликається конструктор базового класу за замовчуванням (без параметрів);

- перед викликом конструктора контейнерного класу (базового або похідного) викликаються конструктори його полів-об'єктів в порядку їх запису у протоколі відповідного класу; якщо ініціалізація об'єктів не здійснюється у списку ініціалізації конструктора контейнерного класу, то викликаються конструктори за замовчуванням.

- викликається конструктор похідного класу.

Деструктори викликаються при виході об'єктів із області досяжності програми або при знищенні об'єктів у динамічній пам'яті. Деструктори успадкованих класів викликаються у зворотному порядку - від похідного до базового класу.

### 1.7. Успадкування статичних даних і методів

За відсутності перекриття об'єкти класів-нащадків мають спільні статичні поля даних, успадковані від A `B::x==A::x`. Якщо у похідному класі B визначити однойменне статичне поле зі статичним полем класу A, то об'єкт класу B матиме спільне з класом A статичне поле `A::x` та індивідуальне статичне поле `B::x!=A::x`.

Статичні методи призначені для роботи зі статичними даними класу. Успадкування статичних методів відбувається за загальними правилами і визначається режимами успадкування. За відсутності перекриття статичні методи базового класу будуть доступними для виклику у похідному класі. Якщо статичний метод похідного класу перекриває статичний метод базового класу, то у похідному класі буде два різні статичні методи.

### 1.8. Успадкування константних елементів класу

Константні елементи класу успадковуються як звичайні елементи з тією відмінністю, що ініціалізація константних елементів класу здійснюється у списку ініціалізації конструктора. Допускається перекриття константних полів. Тоді у похідному класі будуть константні поля, успадковані від базового класу, та власні константні поля.

Константні методи не можуть змінювати значення даних членів класу. Їх успадкування підлягає загальним правилам. Якщо похідний клас перекриває константні методи базового класу, то у похідному класі існують методи, успадковані від базового класу, та власні однойменні методи. Перекриті методи викликаються за допомогою імені базового класу.

```
class A {
protected:
   int x;
public:
 A(int) { this->x=x; }
 int Get() const
        {return x;};
};
```

```
class B: public A {
  int y;
public:
  B(int x, int y):A(x)
     {this->y=y;}
  int Get() const
  {return y;}
};
```

```
void main() {
B b(1,2);
cout<<b.A::Get()<<endl;
cout<<b.Get()<<endl;
}
```

### 1.9. Присвоєння об'єктів при успадкуванні

Об'єкту класу предка можна присвоїти об'єкт одного з класів-нащадків. Це саме справедливе також для вказівників і посилань. Таке присвоєння можливе лише для `public` успадкованих класів.

Зворотне присвоєння об'єкту похідного класу об'єкта базового класу можливе за наявності конструктора за замовчуванням `A::A()` та конструктора перетворення типу `B::B(A)`, наприклад такого: `B:B(A a) {y=a.GetX(); }`

Вказівник (або посилання) на об'єкт похідного класу можна присвоїти вказівнику

(посиланню) з типом базового класу за допомогою відповідного перетворення типів:

```
B* pb=(B*) &a;
B& rb=(B&) a;
```

Таке перетворення не завжди коректне. Для перетворення типів при успадкуванні класів необхідно використовувати операції приведення типів.

**Запитання.**

1. В чому суть успадкування класів. Як оголошується успадкування класів.
2. Як діють режими успадкування класів.
3. Як змінити режим успадкування.
4. Коли виникає перекриття елементів класу. Як доступитися до перекритих елементів.
5. Як при успадкуванні отримати доступ до закритої частини базового класу.
6. Порядок виклику конструкторів та деструкторів при успадкуванні.
7. Як успадковуються статичні методи та дані.
8. Як успадковуються константні елементи класу.
9. Присвоєння об’єктів при успадкуванні.

**Завдання.**

1. Розробити клас ВЕКТОР цілих чисел. Успадкувати цей клас для створення СТЕКА. Визначити необхідні дані, конструктори, деструктор та методи занесення елемента у стек та зчитування елемента зі стеку. Вивести вміст створеного стека на екран.

2. Створити клас КІМНАТА з даними: ширина, дрвжзина, висота. Визначити конструктор і метод доступу. Створити клас ОДНОКІМНАТНОЇ КВАРТИРИ, яка містить кімнату і кухню, номер та поверх. Визначити конструктори, методи доступу. Визначити `public`-похідний клас ОДНОКІМНАТНИХ КВАРТИР РІЗНИХ МІСТ (додатковий параметр - назва міста). Визначити конструктори, деструктор і функцію виведенння.

3. Створити ієрархію класів СПОРТИВНА ГРА і ФУТБОЛ. Перевизначити виведення у потік і введення з потоку, конструктор копіювання, операцію присвоєння через відповідні функції базового класу.

4. Створити клас АВТОМОБІЛЬ, який має марку (вказівник на рядок), колір, об’єм двигуна, потужність, номер реєстрації. Визначити конструктори, деструктор і функцію виведення. Створити public-похідний клас ВАНТАЖІВКА, що має вантажопідйомність кузова. Визначити конструктори без параметрів і з різним числом параметрів, деструктор, методи виведення. Визначити методи перепризначення кольору і номера реєстрації.

5. Створити клас ПРЯМОКУТНИК (довжина, ширина) з методом обчислення його площі. Створити похідний від нього клас ПАРАЛЕЛЕПІПЕД (довжина ширина, висота) з методом обчислення об’єму. Всі дані для створення об’єктів задаються у програмі. Вивести на екран характеристики об’єктів, їх розміри, площу та об’єм.

6. Створититклас КРАПКА, що має координати. Визначити класи ЕЛІПСІВ і КІЛ. Визначити ієрархію типів. Визначити функції виведення, конструктори, деструктори, обчислення площі.

7. Створити ієрархію класів ВЕКТОР та БЕЗПЕЧНИЙ ВЕКТОР з перевіркою виходу індексу за межі одновимірного масиву. Безпечний вектор визначає змінні: нижню і верхню межу (індекси). Перевизначити виведення у потік і введення з потоку, конструктор копіювання, операцію присвоювання через відповідні функції базового класу.

8. Створити класи транспортних засобів: АВТОМОБІЛЬ, ВАНТАЖІВКА, ПАРОПЛАВ і ЛІТАК. Створити з них ієрархію. В основу ієрархії покласти клас ТРАНСПОРТНИЙ ЗАСІБ зі спільними для усіх цих класів елементами. Визначити функції виведення, конструктори і деструктори.

9. Створити клас РІДИНА, що має назву (вказівник на рядок), густину. Визначити конструктори, деструктор і функцію виведення даних. Створити `public`-похідний клас – БЕЗАЛКОГОЛЬНИЙ НАПІЙ, що має колір та ознаку смаку. Визначити конструктори за

замовчуванням і зрізним числом параметрів, деструктори, функцію виведення. Визначити функції зміни густини, кольору та ознаки смаку.

10. Використовуючи ієрархію та успадкування, створити скласи ВІКНА, ВІКНА З ЗАГОЛОВКОМ і ВІКНА З КНОПКОЮ. Визначити необхідні конструктори, деструктори та метод зміни назви заголовку вікна. Зімітувати натиснення кнопки для закривання вікна.

11. Створити ієрархію класів ОСОБА, СТУДЕНТ і СТУДЕНТ-ДИПЛОМНИК. Перевизначити виведення у потік і введення з потоку, визначити конструктор копіювання, операцію присвоєння через відповідні функції базового класу.

12. Створити клас ОСОБА, що має ім’я (вказівник на рядок), вік, вагу. Визначити конструктори, деструктор і функцію виведення. Створити `public`-похідний клас ШКОЛЯР, який має рік навчання. Визначити конструктори за замовчуванням та з різним числом параметрів, деструктори, функцію виведення. Визначити функції перепризначення віку і класу.

13. Створити базовий клас – ЧЕРГА з двома кінцями. Елементи можуть вилучатися і додаватися з будь-якого кінця. Створити похідний клас ЗВИЧАЙНА_ЧЕРГА. Класи повинні мати конструктори, зокрема конструктор копіювання, деструктори, перевантажені функції виведення у потік і введення з потоку.

## 2. Самостійна робота

**/* 1. rozdil 16, listing 1 — рівень доступу до членів базового класу public. Всі відкриті і захищені члени базового класу стають відкритими і захищеними членами похідного класу */**

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

class base {
  int i, j;
public:
  void set(int a, int b) { i=a; j=b; }
  void show() { cout << i << " " << j << "\n"; }
};
class derived : public base {
  int k;
public:
  derived(int x) { k=x; }
  void showk() { cout << k << "\n"; }
};

int main()
{
  derived ob(3);

  ob.set(1, 2); // access member of base
  ob.show(); // access member of base

  ob.showk(); // uses member of derived class
  return 0;
}
```

**1 2**
**3**

**/* 2. listing 2 - рівень доступу до членів базового класу private. Всі відкриті і захищені члени базового класу стають закритими членами базового класу */**

```
// This program won't compile.
#include <iostream>
using namespace std;

class base {
  int i, j;
public:
  void set(int a, int b) { i=a; j=b; }
  void show() { cout << i << " " << j << "\n";}
};

// Public elements of base are private in derived.
class derived : private base {
  int k;
public:
  derived(int x) { k=x; }
  void showk() { cout << k << "\n"; }
};

int main()
{
  derived ob(3);

  ob.set(1, 2); // error, can't access set()
  ob.show(); // error, can't access show()
```

```
  return 0;
}
```
**c16_2.cpp:9: error: ‘void base::set(int, int)’ is inaccessible**
**c16_2.cpp:25: error: within this context**
**c16_2.cpp:25: error: ‘base’ is not an accessible base of ‘derived’**
**c16_2.cpp:10: error: ‘void base::show()’ is inaccessible**
**c16_2.cpp:26: error: within this context**

**/* 3. listing 3 - рівень доступу до членів базового класу public. Захищені члени базового класу стають захищеними членами похідного класу */**

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

class base {
protected:
  int i, j; // private to base, but accessible by derived
public:
  void set(int a, int b) { i=a; j=b; }
  void show() { cout << i << " " << j << "\n"; }
};

class derived : public base {
  int k;
public:
  // derived may access base's i and j
  void setk() { k=i*j; }

  void showk() { cout << k << "\n"; }
};

int main()
{
  derived ob;

  ob.set(2, 3); // OK, known to derived
  ob.show(); // OK, known to derived

  ob.setk();
  ob.showk();

  return 0;
}
```
**2 3**
**6**

**/* 4. listing 4 — відкрите успадкування захищених членів базового класу похідними класами*/**

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

class base {
protected:
  int i, j;
public:
  void set(int a, int b) { i=a; j=b; }
  void show() { cout << i << " " << j << "\n"; }
};

// i,j успадковуються, як захищені члени.
class derived1 : public base {
  int k;
public:
  void setk() { k = i*j; } // legal
  void showk() { cout << k << "\n"; }
```

```
};

// i, j успадковуються, як захищені члени через успадкований клас derived1.
class derived2 : public derived1 {
  int m;
public:
  void setm() { m = i-j; } // legal
  void showm() { cout << m << "\n"; }
};

int main()
{
  derived1 ob1;
  derived2 ob2;

  ob1.set(2, 3);
  ob1.show();
  ob1.setk();
  ob1.showk();

  ob2.set(3, 4);
  ob2.show();
  ob2.setk();
  ob2.setm();
  ob2.showk();
  ob2.showm();

  return 0;
}
2 3
6
3 4
12
-1

/* 5. listing 5 -  закрите успадкування базового класу */
// Програма не компілюється
#include <iostream>
using namespace std;

class base {
protected:
  int i, j;
public:
  void set(int a, int b) { i=a; j=b; }
  void show() { cout << i << " " << j << "\n"; }
};

// Всі елементи базового класу є закритими членами класу derived1.
class derived1 : private base {
  int k;
public:
  // це можна робити, оскільки i, j є закритими членами класу derived1
  void setk() { k = i*j; } // OK
  void showk() { cout << k << "\n"; }
};

// доступ до членів i, j, set(), and show() не успадковуэться
class derived2 : public derived1 {
  int m;
public:
  // нема доступу до i,j, оскільки i, j є  закритими членами derived1
  void setm() { m = i-j; } // Помилка
  void showm() { cout << m << "\n"; }
```

```
};

int main()
{
  derived1 ob1;
  derived2 ob2;
  ob1.set(1, 2); // помилка, не можна викликати функцію set()
  ob1.show(); // помилка, не можна викликати функцію show()
  ob2.set(3, 4); // помилка, не можна викликати функцію set()
  ob2.show(); // помилка, не можна викликати функцію show()
  return 0;
}
```

```
c16_5.cpp:8: error: ‘int base::i’ is protected
c16_5.cpp:28: error: within this context
c16_5.cpp:8: error: ‘int base::j’ is protected
c16_5.cpp:28: error: within this context
c16_5.cpp: In function ‘int main()’:
c16_5.cpp:10: error: ‘void base::set(int, int)’ is inaccessible
c16_5.cpp:37: error: within this context
c16_5.cpp:37: error: ‘base’ is not an accessible base of ‘derived1’
c16_5.cpp:11: error: ‘void base::show()’ is inaccessible
c16_5.cpp:38: error: within this context
```

```
/* 6. listing 6 — захищене успадкування */
#include <iostream>
using namespace std;

class base {
protected:
  int i, j; // закриті члени класу base, доступні класу derived
public:
  void setij(int a, int b) { i=a; j=b; }
  void showij() { cout << i << " " << j << "\n"; }
};

// Клас, отриманий за допомогою захищеного успадкування
class derived : protected base{
  int k;
public:
  // клас derived має доступ до членів i, j і setij() з класу base.
  void setk() { setij(10, 12); k = i*j; }

  // звідси можна викликати функцію showij()
  void showall() { cout << k << " "; showij(); }
};

int main()
{
  derived ob;

//  ob.setij(2, 3); // невірно, setij() є захищений член класу derived

  ob.setk(); // вірно, викликається відкритий член класу derived
  ob.showall(); // вірно, викликається відкритий член класу derived

// ob.showij(); // невірно, функція showij() є захищени членом класу derived
  return 0;
}
```

```
120 10 12
```

```
/* 7. listing 8 — послідовність виклику конструкторів і деструкторів */
#include <iostream>
using namespace std;
```

```
class base {
public:
  base() { cout << "Створюється об'єкт класу base\n"; }
  ~base() { cout << "Знищується об'єкт класу base\n"; }
};

class derived: public base {
public:
  derived() { cout << "Створюється об'єкт класу derived\n"; }
  ~derived() { cout << "Знищується об'єкт класу  derived\n"; }
};

int main()
{
  derived ob;

  // тільки створюється і знищується об'єкт

  return 0;
}
```

**Створюється об'єкт класу base**
**Створюється об'єкт класу derived**
**Знищується об'єкт класу  derived**
**Знищується об'єкт класу base**

```
/* 8. listing 10 — виклик конструкторів при ієрархічному успадкуванні */
#include <iostream>
using namespace std;

class base {
public:
  base() { cout << "Створення базового класу base\n"; }
  ~base() { cout << "Знищення базового класу base\n"; }
};

class derived1 : public base {
public:
  derived1() { cout << "Створення базового класу derived1\n"; }
  ~derived1() { cout << "Знищення базового класу derived1\n"; }
};

class derived2: public derived1 {
public:
  derived2() { cout << "Створення базового класу derived2\n"; }
  ~derived2() { cout << "Знищення базового класу derived2\n"; }
};

int main()
{
  derived2 ob;

  // construct and destruct ob

  return 0;
}
```

**Створення базового класу base**
**Створення базового класу derived1**
**Створення базового класу derived2**
**Знищення базового класу derived2**
**Знищення базового класу derived1**
**Знищення базового класу base**

```
/* 9. listing 12 -  виклик конструкторів при множинному успадкуванні */
```

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

class base1 {
public:
  base1() { cout << "Створення об'єкта класу  base1\n"; }
  ~base1() { cout << "Знищення об'єкта класу  base1\n"; }
};

class base2 {
public:
  base2() { cout << "Створення об'єкта класу base2\n"; }
  ~base2() { cout << "Знищення об'єкта класу base2\n"; }
};

class derived: public base1, public base2 {
public:
  derived() { cout << "Створення о'бєкта класу derived\n"; }
  ~derived() { cout << "Знищення об'єкта класу derived\n"; }
};

int main()
{
  derived ob;
  // створення і знищенння об'єкта ob
  return 0;
}
```

**Створення об'єкта класу  base1**
**Створення об'єкта класу base2**
**Створення о'бєкта класу derived**
**Знищення об'єкта класу derived**
**Знищення об'єкта класу base2**
**Знищення об'єкта класу  base1**

```
/* 10. listing 16 — передача параметрів конструктору базового класу */
#include <iostream>
using namespace std;

class base {
protected:
  int i;
public:
  base(int x) { i=x; cout << "Створення об'єкта класу base\n"; }
  ~base() { cout << "Знищення об'єкта класу base\n"; }
};

class derived: public base {
  int j;
public:
  // клас derived використовує x; а y передається класу base
  derived(int x, int y): base(y)
    { j=x; cout << "Створення об'єкта класу derived\n"; }

  ~derived() { cout << "Знищення об'єкта класу derived\n"; }
  void show() { cout << i << " " << j << "\n"; }
};

int main()
{
  derived ob(3, 4);
  ob.show(); // висвічує 4 3
  return 0;
}
```

**Створення об'єкта класу base**

```
Створення об'єкта класу derived
4 3
Знищення об'єкта класу derived
Знищення об'єкта класу base
```

```
/* 11. listing 17 - передача параметрів конструкторам базових класів при множинному успадкуванні */
#include <iostream>
using namespace std;

class base1 {
protected:
  int i;
public:
  base1(int x) { i=x; cout << "Створення об'єкту base1\n"; }
  ~base1() { cout << "Знищення об'єкту base1\n"; }
};

class base2 {
protected:
  int k;
public:
  base2(int x) { k=x; cout << "Створення об'єкту base2\n"; }
  ~base2() { cout << "Знищення об'єкту base1\n"; }
};

class derived: public base1, public base2 {
  int j;
public:
  derived(int x, int y, int z): base1(y), base2(z)
    { j=x; cout << "Створення об'єкту derived\n"; }

  ~derived() { cout << "Знищення об'єкту derived\n"; }
  void show() { cout << i << " " << j << " " << k << "\n"; }
};

int main()
{
  derived ob(3, 4, 5);

  ob.show(); // висвічує 4 3 5

  return 0;
}
```

```
Створення об'єкту base1
Створення об'єкту base2
Створення об'єкту derived
4 3 5
Знищення об'єкту derived
Знищення об'єкту base1
Знищення об'єкту base1
```

```
/* 12. listing 18 — конструктор похідного класу не має власних параметрів */
#include <iostream>
using namespace std;

class base1 {
protected:
  int i;
public:
  base1(int x) { i=x; cout << "Створення об'єкту base1\n"; }
  ~base1() { cout << "Знищення об'єкту base1\n"; }
};
```

```
class base2 {
protected:
  int k;
public:
  base2(int x) { k=x; cout << "Створення об'єкту base2\n"; }
  ~base2() { cout << "Знищення об'єкту base2\n"; }
};

class derived: public base1, public base2 {
public:
  /* Конструктор класу Derived не має параметрів */

  derived(int x, int y): base1(x), base2(y)
    { cout << "Створення об'єкту derived\n"; }

  ~derived() { cout << "Знищення об'єкту derived\n"; }
  void show() { cout << i << " " << k << "\n"; }
};

int main()
{
  derived ob(3, 4);

  ob.show(); // висвічує 3 4

  return 0;
}
```

**Створення об'єкту base1**
**Створення об'єкту base2**
**Створення об'єкту derived**
**3 4**
**Знищення об'єкту derived**
**Знищення об'єкту base2**
**Знищення об'єкту base1**

```
/* 13. listing 22 — відновлення закритого статусу закритих членів */
#include <iostream>
using namespace std;

class base {
  int i; // закритий член класу base
public:
  int j, k;
  void seti(int x) { i = x; }
  int geti() { return i; }
};

// закрите успадкування класу  base
class derived: private base {
public:
  /* Відновлення відкритого статусу членів
     j,  seti(), geti() */
  base::j;    // змінна j відкрита, а k  - ні
  base::seti; // функція seti() відкрита
  base::geti; // функція geti() відкрита

// base::i; // невірно, рівень доступу піднімати не можна

  int a; // відкритий член
};

int main()
{
  derived ob;
```

```
//ob.i = 10; // невірно, так як змінна i є закритим членом класу derived

  ob.j = 20; // невірно, так як змінна j відкрита в класі derived
//ob.k = 30; // невірно, так як змінна k є закритим членом в класі  derived

  ob.a = 40; // вірно, так як a є відкритим членом класа derived
  ob.seti(10);

  cout << ob.geti() << " " << ob.j << " " << ob.a;

  return 0;
}
```

**10 20 40**

**/* 14. listing 25 — застосування оператора розкриття області видимості для усунення неоднозначності для доступу до членів базового класу */**

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

class base {
public:
  int i;
};

// клас derived1 успадковує клас  base
class derived1 :  public base {
public:
  int j;
};

// клас derived2 успадковує клас  base
class derived2 : public base {
public:
  int k;
};

/* клас derived3 успадковує класи  derived1 і derived2 і має дві копії класу base
*/
class derived3 : public derived1, public derived2 {
public:
  int sum;
};

int main()
{
  derived3 ob;

  ob.derived1::i = 10; // для усунення неоднозначності використовується змінна i з
класу derived1
  ob.j = 20;
  ob.k = 30;

  // неоднозначність усунена
  ob.sum = ob.derived1::i + ob.j + ob.k;

  // неоднозначність усунена
  cout << ob.derived1::i << " ";

  cout << ob.j << " " << ob.k << " ";
  cout << ob.sum;

  return 0;
}
```

**10 20 30 60**

```
/* string 26  використання віртуального базового класу  */
#include <iostream>
using namespace std;

class base {
public:
  int i;
};

// клас derived1 успадковує віртуальний клас base
class derived1 : virtual public base {
public:
  int j;
};

// клас derived2 успадковує віртуальний клас base
class derived2 : virtual public base {
public:
  int k;
};

/* клас derived3 успадковує класи  derived1 і derived2.
   Але в цьому випадку він має одну копію базового класу */
class derived3 : public derived1, public derived2 {
public:
  int sum;
};

int main()
{
  derived3 ob;

  ob.i = 10; // неоднозначність відсутня
  ob.j = 20;
  ob.k = 30;

  // неоднозначність відсутня
  ob.sum = ob.i + ob.j + ob.k;
  // неоднозначність відсутня
  cout << ob.i << " ";
  cout << ob.j << " " << ob.k << " ";
  cout << ob.sum;

  return 0;
}
```

**10 20 30 60**