**Лабораторна робота №2**

**Розробка програм з використанням принципів успадкування.**

**Мета роботи**: Набуття навичок розробки програм з використанням базового принципу об’єктно-орієнтованого програмування – успадкування. Використання конструкторів та деструкторів при наслідуванні.

**Загальні вказівки**

Лабораторна робота складається з двох завдань.

1. В першому завданні потрібно створити базовий та похідний клас згідно варіанту (просте наслідування).
2. В другому завданні потрібно реалізувати множинне наслідування.

Для перевірки надається звіт та файли з текстом розроблених програм

Структура звіту:

1. Титульний аркуш

2. Текст завдання.

3. Хід виконання завдання з детальними поясненнями, що включають:

3.1. Опис вхідних та вихідних даних; описовий алгоритм розв'язання завдання з доцільними поясненнями; схема алгоритму розв'язання завдання.

3.2. Лістинги програм.

3.3. Розробка тестів та результати тестування. Скріншоти вхідних та результатів розв'язання завдання на ЕОМ.

3.5. Аналіз отриманих результатів.

4. Висновки по роботі.

**Теоретичні відомості**

1. Конструктори та деструктори

Зазвичай у найпростіших програмах існує два способи визначення цілочисельної змінної:

1) можна визначити змінну, а потім присвоїти її деяке значення, наприклад:

іnt weіght;

weіght = 5;

2) можна визначити змінну і негайно її ініціалізувати, наприклад:

іnt w = 7;

при цьому операція ініціалізації сполучить у собі визначення змінної з присвоєнням початкового значення.

Як же ініціалізувати змінні-члени класу ?

Для цього в ООП використовуються дві спеціальні функції:

1) Конструктор - це метод класу, ім'я якого збігається з ім'ям класу, призначений для створення й ініціалізації - класу.

2) Деструктор - це метод класу, ім'я якого складається з ( ~ )- тільди й імені класу, призначений для видалення з пам'яті об'єкти класу, що відробили, і звільнення виділеної для них пам'яті.

Наприклад, для класу CTіme ім'я деструктора ***~*** CTіme ( );

**Правила роботи з конструкторами і деструкторами**

1. У класах може бути оголошено декілька конструкторів для автоматичної ініціалізації об'єкта класу при його створенні.

2. У класах може бути оголошений тільки один деструктор, викликаємий для очищення пам'яті при виході об'єкта класу з області видимості.

3. Конструктори і деструктори схожі на звичайні функції-члени, але вони рідше викликаються безпосередньо в операторах програми.

4. С++ автоматично викликає конструктори і деструктори для ініціалізації й очищення об'єктів класу.

5. Конструктори з'являються як функції члени, що не повертають ніяких значень, з довільним числом параметрів будь-якого типу.

6. Деструктори не приймають ніяких аргументів і не повертають ніяких значень.

7. Конструктори, як правило, з'являються у відкритій секції.

8. Якщо конструктор оголошений без параметрів, він викликається за замовчуванням.

9. Деструктор викликається автоматично, коли об'єкт класу виходить з області видимості.

10. Конструктор викликається автоматично в момент створення об'єкта чи класу за бажанням програміста.

Приклад використання конструктора і деструктора

# include < iostream.h >

# include < time.h >

# include < string.h >

class CTime

{

private:

long dt*;* // Дата і час в секундах від 1 січня 1970 року.

Схована сhar \* dts; // Представлення (містить адреси) дати область у вигляді рядка

*voіd Delete Dts ( voіd );*

// видаляє показник dts; *закрита функція*

рublic:

CTime (); // Конструктор *1 –* викликається за замовченням

СTime (int m,int d=-1, int y=-1, int h2=-1, int min=-1);

// Конструктор 2 - перегружений

~ CTime (); // Деструктор

void Display(void) {cout<<ctime(&dt;}

void GetTime(int &m,int &d,int &y, int &h2, int &min);

void SetTime(int m=-1,int d=-1,int y=-1,int h2=- 1,int min=-1);

const char\*Get Stime (void);

void Change Time (long n minuts)

{dt+=(n minutes×60);Delete Dts();}

};

Обов'язки типового конструктора обмежені присвоюванням початкових значень даним-членам класу, виділенням пам'яті, використаної об'єктом класу і т.д. С++ автоматично викликає конструктор для ініціалізації об'єкта класу CТіme, отже, усі такі об'єкти гарантовано мають ініціалізований показник dts.

Мета конструктору - забезпечити ясно визначену ініціалізацію об'єктів класу.

Приклад використання конструкторів

main ( )

{

СТime t1;

СТime t2 (8);

СТime t3 (8, 1);

СТime t4 (8, 1, 1999);

СТime t5 (8, 1, 1999, 8);

СТime t6 (8, 1, 1999, 8, 30);

t1. Display ();

t2. Display ();

………………

t6. Display ();

return ();}

**Приклад використання деструктора**

*int main ( )*

*{*

*func ();*

*return ();*

*}*

*void func (void)*

{

// Конструктор викликається автоматично за замовчуванням щораз при виклику функції

CTime today;

Const char \* sp;

//Pезультат функції зберігається в локальному символьному показнику *sp*

sp = today. Get STime ()*;*

Cout << “ First time: “ << sp;

sp = today. Get STime ();

Cout << “ Second time: “ << sp;

}

Об'єкт класу Сtіme todayоголошений в середині класу, тобто він локальний в області видимості функції funс. При описанні функцій-членів класу деструктор може мати вигляд:

CTime:: ~ CTime ( ); - Деструктор

{

delete dts;

}

Const char \* CTime :: Get STime (void)

{

//перевірка чи не містить *dts* шлях *return dts;* інакше викликаються функції *Strdup ( ) і CTіme ( )* для перетворення поточної дати і часу в рядок зі збереженням адреси рядка в символьному покажчику *dts* - закритому члені класу.

if (dts)

dts = strdup (CTime (&dt));

return dts;

}

Тоді наступні виклики GetSTіme() будуть використовувати раніше створений рядок. Це скорочує кількість виділень простору в області, що динамічно розподіляється, і прискорює швидкодію програми. Клас СТіme володіє показником*dts*, що посилається на виділений блок пам'яті, та керує розподілом цієї пам'яті. При виході об'єкта класу з області видимості, область пам'яті звільняється деструктором автоматично.

**2. Успадкування (наслідування)**

Для створення класів з доданою функціональністю вводять успадкування. Клас-нащадок має поля і функції-члени базового класу, але не має права звертатися до приватних (private) полів і функцій базового класу. У цьому і полягає різниця між приватними і захищеними членами.

Клас-нащадок може додавати свої поля і функції або перевизначати функції базового класу.

За умовчанням, конструктор нащадка без параметрів викликає конструктор базового класу, а потім конструктори доданих елементів.

Деструктор працює в зворотному порядку. Інші конструктори доводиться визначати кожного разу наново.

**Приклад 1**

class ArrayWithAdd : public Array {

ArrayWithAdd(int n) : Array(n) {}

ArrayWithAdd() : Array() {}

ArrayWithAdd(const Array& a) : Array(a) {}

void Add(const Array& a);

};

Нащадок – це більш ніж базовий клас, тому він може використовуватися скрізь, де використовується базовий клас, але не навпаки.

Успадкування буває публічним, захищеним і власним. При публічному спадкуванні, публічні і захищені члени базового класу зберігають свій статус, а до приватних не можуть звертатися навіть функції-члени нащадка. Захищене спадкування відрізняється тим, що при ньому публічні члени базового класу є захищеними членами нащадка. При приватному успадкуванні, до жодного члена базового класу навіть функції-члени нащадка права звертатися не мають. Як правило, публічне спадкування зустрічається значно частіше за інші.

Клас може бути нащадком декількох класів. Це називається *множинним спадкуванням*. Такий клас володіє полями і функціями-членами всіх його предків. Наприклад, клас FlyingCat може бути нащадком класів Cat і FlyingAnimal.

**Приклад 2**

class Cat {

...

void Purr();

...

};

class FlyingAnimal {

...

void Fly();

...

};

class FlyingCat : public Cat, public FlyingAnimal {

...

PurrAndFly() {Purr(); Fly();}

...

};

Існує два типи успадкування. Це одиночне і множинне. Суть одиночного полягає в тому, що дочірній клас наслідує дані і властивості тільки одного батьківського класу. При множинному успадкування кількість батьківських класів повинна бути не меншою за один. При множинному успадкуванні можуть виникати проблеми із тим, що у батьківських класів є методи або атрибути із однаковими назвами. При їх використанні слід явно вказувати від якого з батьківських класів використовується метод або атрибут. Зараз множинне успадкування вже майже не використовується і всі нові мови програмування відмовились від цього принципу. Із популярних мов програмування множинне успадкування підтримує лише С++.

***Керування доступом при успадкуванні***

Повернемося до синтаксису успадкування. Специфікатори доступу – public, private, protected при призначенні типу успадкування можуть пропускатися, при цьому керуються наступним правилом: якщо визначається class, то по замовчуванню похідний клас приймається як private.

Наступна таблиця містить визначення рівня доступу в середині похідного класу. У першій колонці – специфікатор доступу, що визначає успадкування між класами, у двох подальших – рівень доступу у базовому та похідному класах:

|  |  |  |
| --- | --- | --- |
| Тип успадкування  class В: [] class А | Доступ у базовому класі А | Доступ у похідному класі В |
| public | private  public  protected | недоступно  public  protected |
| private | private  public  protected | недоступно  private  private |
| protected | private  public  protected | недоступно  protected  protected |

З цієї таблиці, видно, які можливості надає механізм успадкування. Так, при відкритому успадкуванні public загальнодоступні та захищені елементи-дані зберігають свої рівні доступу надалі, і лише рrivate-елементи виявляються недоступними вниз по ієрархії.

Слід дотримуватися наступних правил успадкування методів у похідному класі:

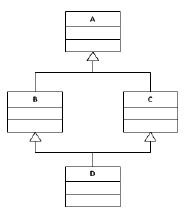
1. Оскільки конструктори не успадковуються, похідні класи повинні мати власні конструктори. Тут можуть бути дві ситуації:

• якщо у конструкторі похідного класу відсутній явний виклик конструктора базового класу, автоматично викликається конструктор базового класу по замовчуванню (той, що не має параметрів). Для ієрархії декількох рівнів конструктори базових класів викликаються, починаючи з найвищого рівня.

• якщо конструктор базового класу потребує вказівку параметрів, він повинен бути явно викликаний в конструкторі похідного класу списком ініціалізації.

2. Оскільки деструктор не успадковуються та програмою не визначений деструктор у похідному класі, його буде згенеровано по замовчуванню і через нього викликано деструктори усіх базових класів. У класовій ієрархії деструктори викликаються у порядку, зворотному до виклику конструкторів; спочатку деструктор поточного класу, а потім деструктор базового класу.

3. Похідний клас може перевизначати метод з одним і тим же ім'ям, що і у базовому класі, відповідно коректуючи його поведінку для себе. Аби запобігти неоднозначностям, рекомендовано перевизначати лише віртуальні методи класів.



Ситуація, зображена на рисунку можлива тільки для мови С++, оскільки вона допускає прояв множинного успадкування. Для того, щоб не виникало проблем, необхідно описувати успадкування класів В і С, як віртуальне від класу А. Тоді при створення об‘єктів класу D не буде дублюватися клас А.

**Приклад 3**

class **FirstClass**    // базовий клас

{

protected:          // специфікатор доступу до елементу value

    int value;

public:

    FirstClass()       {  value = 0;   }

    FirstClass(int x)  {  value = x;   }

    void show\_value()  {  cout << value << endl;  }

};

class **SecondClass** : public FirstClass   // похідний клас

{

public:

          // конструктор класу SecondClass викликає конструктори класу FirstClass

    SecondClass() : FirstClass() {}

    SecondClass(int inputS) : FirstClass(inputS) {}

    void ValueSqr() // Без специфікатора  protected не змогли б змінити value

    {

        value \*= value;

    }

};

int main()

{   setlocale(LC\_ALL, "rus");

    FirstClass F\_object(3);     // об’єкт базового класу

    cout << "\n\tvalue F\_object = ";

    F\_object.show\_value();

    SecondClass S\_object(4);    // об’єкт похідного класу

    cout << "\tvalue S\_object = ";

    S\_object.show\_value();    // виклик методу базового класу

    S\_object.ValueSqr();      // підносимо value до квадрату

    cout << "\tквадрат value S\_object = ";

    S\_object.show\_value();

    // F\_object.ValueSqr();      // ПОМИЛКА: немає доступу

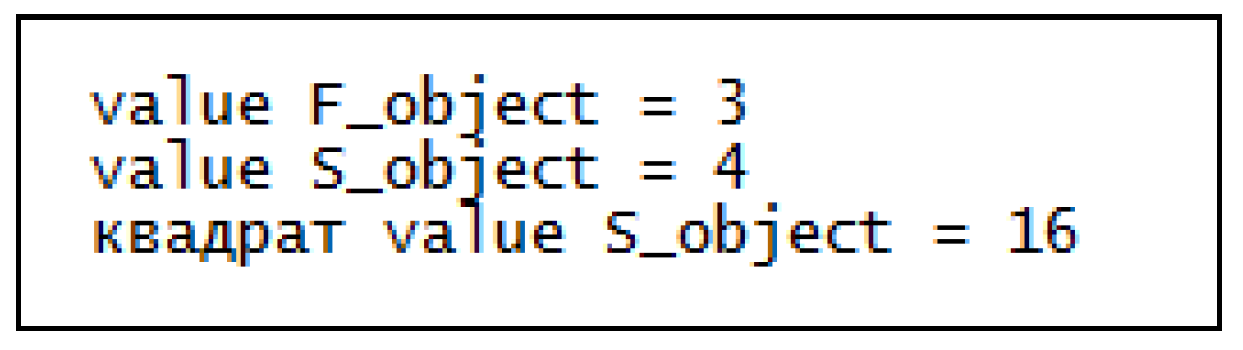
    cout << endl;

    system("pause");

    return 0;

}

Результат роботи програми



**Завдання на виконання роботи**

***Завдання 1***

**Завдання 1.1**  Створити клас для обробки записів бази даних у відповід­ності з вказаною предметною областю. Розробити програму, дотримуючись таких вимог:

1. Реалізувати окремими функціями класу:
   * конструктори без параметрів та з параметрами;
   * деструктор;
   * виведення інформації на екран;
   * редагування записів.
2. При розробці програми слід здійснити захищення даних (опис з модифікатором private) для ізоляції елементів-даних класу від підпрограм, в яких цей клас використовується.
3. В головній функції передбачити створення масиву об’єктів класу та виконання пошуку згідно вказаного критерію.

**Варіанти індивідуальних завдань**

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Предметна область** | **Реквізити об’єкту** | **Параметр пошуку** |
| 1. | Бібліотека | Інвентарний номер, автор, назва, кількість сторінок, рік видання | Автор |
| 2. | Телефонний довідник | Прізвище, ім'я, по батькові, домашня адреса, телефон. | Прізвище |
| 3. | Розклад руху літаків | Номер рейсу, тип літака, напрямок руху, періодичність вильоту. | Тип літака |
| 4. | Колекція компакт-дисків | Інвентарний номер, назва альбому, об'єм диску, тип, дата запису. | Назва альбому |
| 5. | Записна книжка | Прізвище, ім'я, домашня адреса, телефон, електронна пошта. | Електронна пошта |
| 6. | Предметний покажчик | Слово; номера сторінок, де це слово зустрічається; кількість цих слів на даній сторінці | Слово |
| 7. | Розклад пар | Номер пари, предмет, прізвище викладача, форма заняття. | Номер пари |
| 8. | Список файлів | ім'я файла, розширення, розмір, дата створення, атрибути. | Дата створення |
| 9. | Архів програм | Назва програми, операційна система, розмір програми, дата запису | Операційна система |
| 10. | Рахунки банку | Прізвище, ім'я, дата останньої операції, сума вкладу | Дата операції |
| 11. | Користувачі локальної мережі | Прізвище, група, обліковий запис, тип облікового запису. | Прізвище |
| 12. | Камера схову | Прізвище, дата здачі, термін зберігання, інвентарний номер та назва предмета | Дата здачі |
| 13. | Склад товарів | Інвентарний номер, назва товару, вага, ціна, кількість | Назва товару |
| 14. | Каса продажу квитків | Назва пункту, час відправлення, дата відправлення, час прибуття, дата прибуття, ціна квитка | Назва пункту |
| 15. | Успішність студентів | Прізвище, номер групи, оцінки з трьох предметів | Номер групи |
| 16. | Мобільний оператор | Прізвище, телефонний номер, тариф та вартість тарифу, сума на рахунку | Тариф |
| 17. | Компанія-перевізник | Назва транспорту, тип, швидкість, витрати на пальне, пробіг | Тип транспорту |
| 18. | Пункт прокату велосипедів | Прізвище, номер паспорта, номер телефону, марка велосипеда, взятого на прокат, термін прокату | Марка велосипеда |
| 19. | Поліклініка | Прізвище лікаря, спеціальність, кабінет, години прийому | Спеціальність |
| 20. | Лікарня | Прізвище пацієнта, адреса, номер телефону, номер медичної карти, діагноз | Діагноз |
| 21. | Туристична фірма | Країна, готель, тривалість туру, вартість обслуговування одного клієнта | Готель |
| 22. | Податкова служба | Назва підприємства, рівень оподаткування, рік заснування, власник | Рівень оподаткування |
| 23. | Клінінгова компанія | Фірма-замовник, час останнього прибирання, інтенсивність використання приміщення, площа приміщення | Площа приміщення |
| 24. | Колекція артефактів | Назва, місце походження, рік знаходження, прізвище колекціонера | Місце походження |
| 25. | Колекція гравця | Назва гри, вартість гри, жанр, фірма-виробник | Жанр |
| 26. | Іграшки | Ціна, назва, кількість на складі, фірма-виробник | Фірма-виробник |
| 27. | Агенція нерухомості | Будівля, адреса, площа, поверх, прізвище власника | Поверх |
| 28. | Хімічні елементи | Назва, атомний номер, масове число, валентність | Валентність |
| 29. | Флот | Назва корабля, порт приписки, водотоннажність, рік спуску | Порт приписки |
| 30. | Книжкова продукція | Назва, видавництво, рік видання, ціна, кількість сторінок | Видавництво |
| 31. | Кулінарний інструмент | Назва, матеріал виготовлення, вага, фірма-випрбник | Матеріал виготовлення |
| 32. | Колекція музею | Назва, рік виготовлення, автор роботи, країна | Країна |

**Завдання 1.2.** Утворити похідний клас, залучивши до нього як мінімум два додаткових поля таким чином, щоб клас набув більшої спеціалізованості. Для похідного класу використати кон­структор, щоб він містив усі аргументи, необхідні для ініціалізації об'єкту похідного класу. Створити додаткові необхідні функції, що дозволяють перевірити роботу похідних класів.

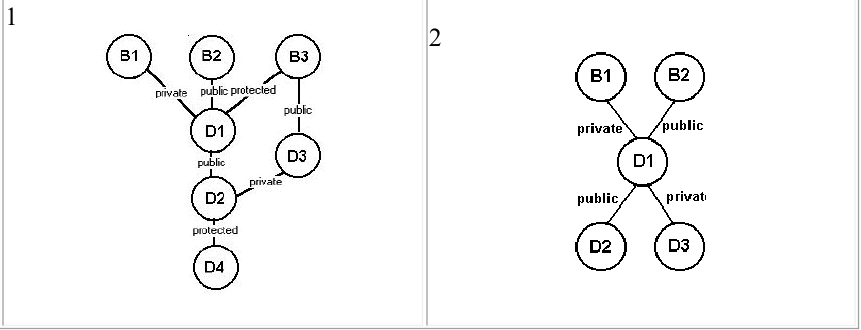
***Завдання 2***

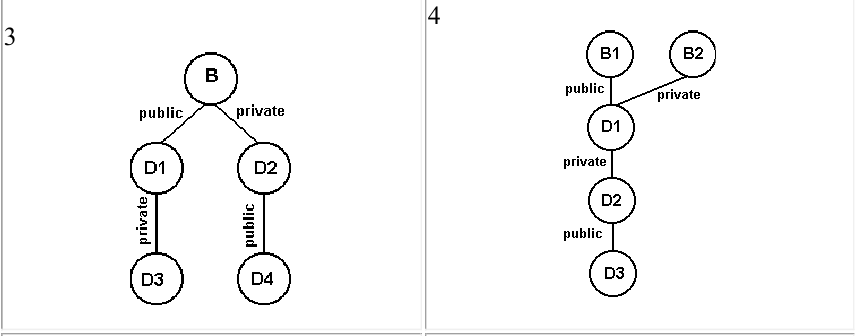
Необхідно побудувати ієрархію класів відповідно схемі успадкування, наведеній у варіанті завдання.

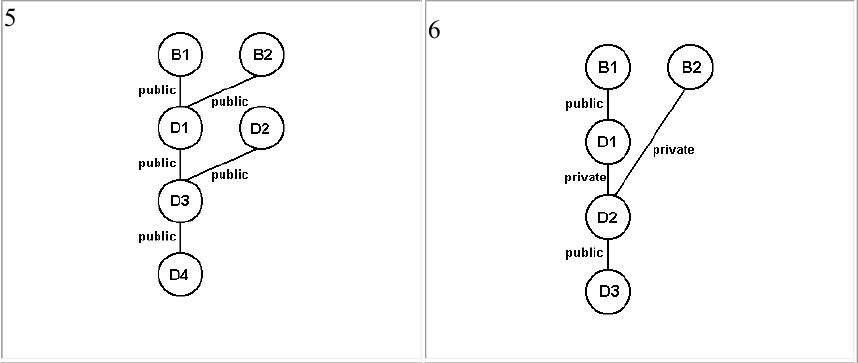
Кожен клас повинен містити конструктор ініціалізації, і функцію show для виведення значень.

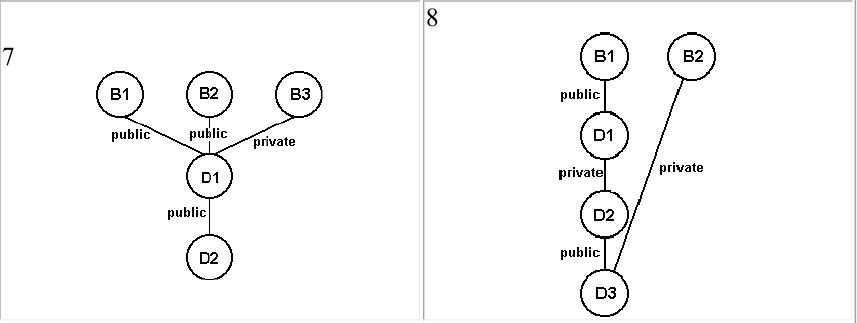
Функція main повинна ілюструвати ієрархію успадкування.

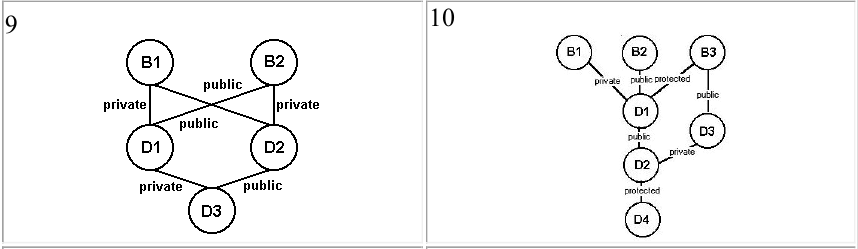
**Варіанти індивідуальних завдань**

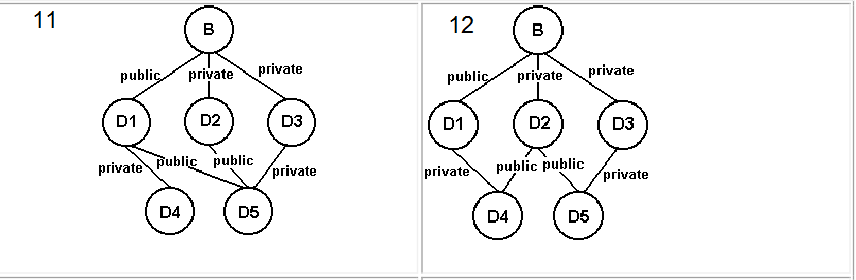


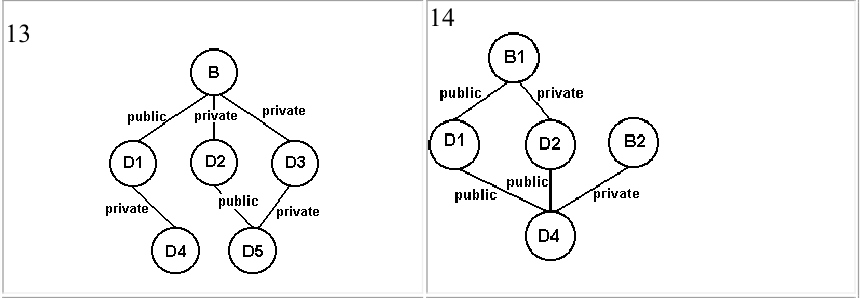


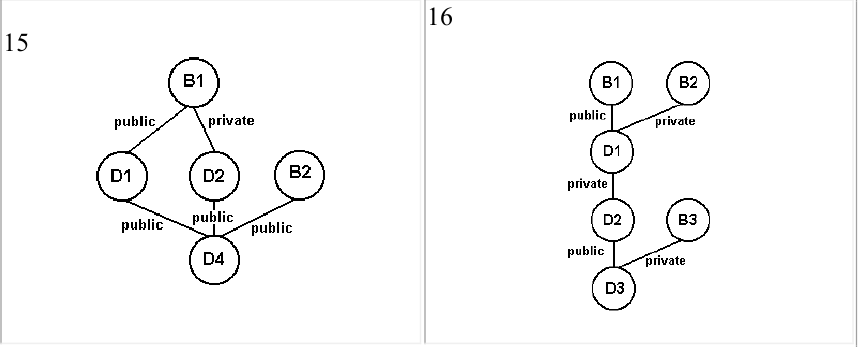


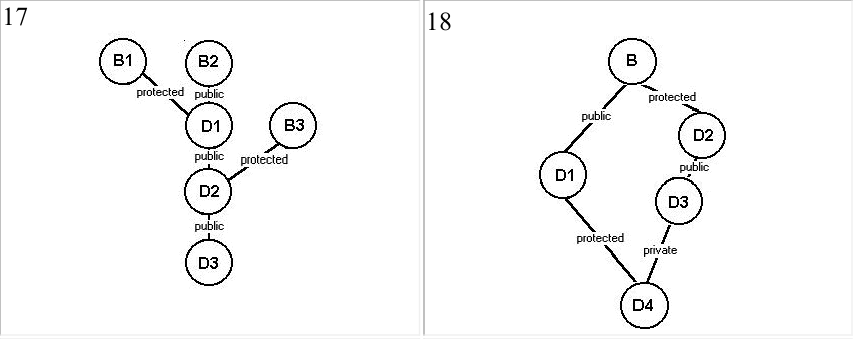


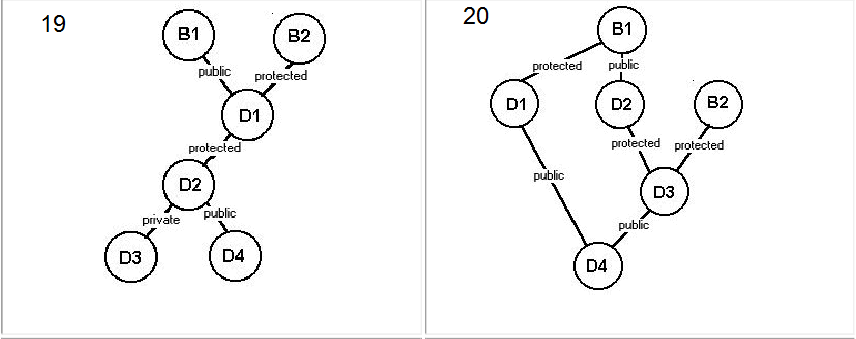


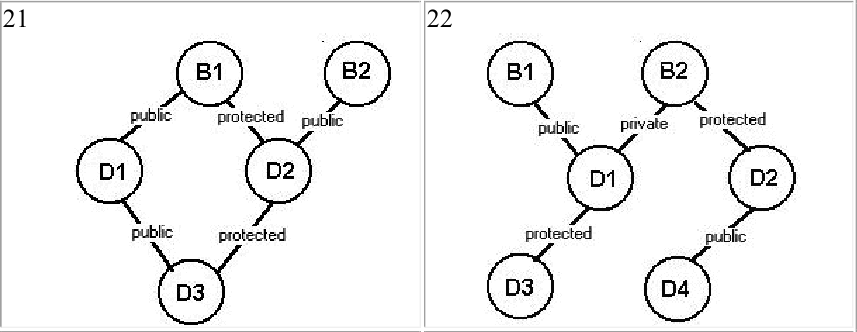


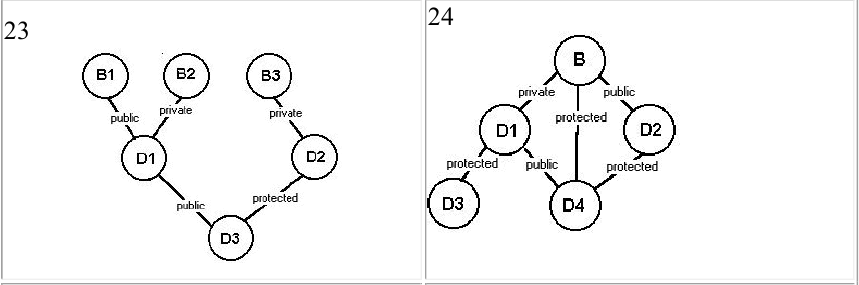


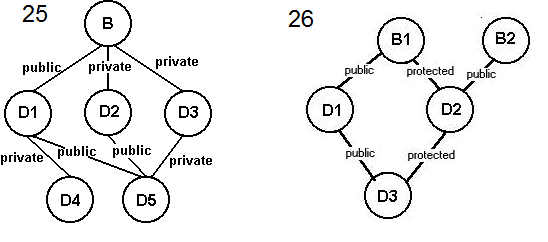


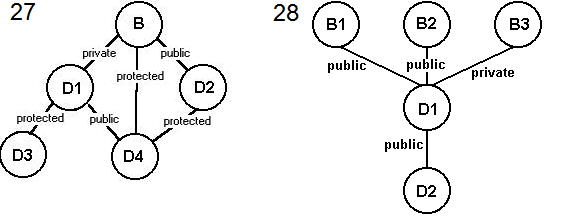


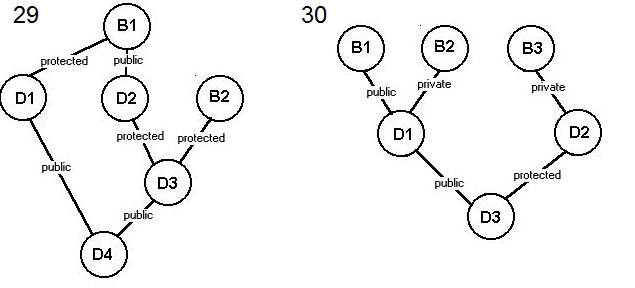


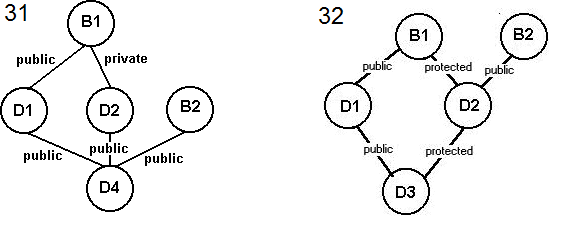




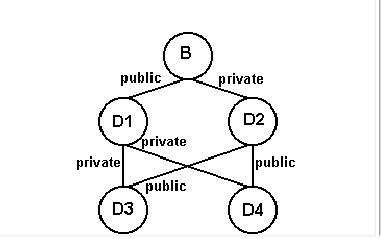








**Приклад виконання завдання 2.**



Спочатку створимо необхідну ієрархію класів:

class B{

int a;

public:

};

class D1: public B {

int b;

public:

};

class D2: private B{

int c;

public:

};

class D3: private D1, public D2 {

int d;

public:

};

class D4: public D2, private D1 {

int e;

public:

};

Створимо у всіх класах конструктори, які змогли б по ланцюгу успадкування ініціалізувати свої змінні і передати решту значень далі до конструктора базового класу.

B(int x) { a=x; }

D1(int x, int y) : B(y) { b=x;};

D2(int x, int y) : B(y) { c=x;};

D3(int x, int y, int z, int i, int j) : D1(y,z), D2(i,j) { d=x;}

D4(int x, int y, int z, int i, int j) : D1(y,z), D2(i,j) { e=x;}

Тепер добавимо в кожен клас функцію show, яка б виводила на екран змінну із розділу private класу, якому належить сама і викликала функцію show тих класів, які стоять вище по ієрархії успадкування.

void show\_B() { cout <<"B= "<< a << "\n"; }

void show\_D1() { cout << "D1= "<< b << "\n"; show\_B();}

void show\_D2() { cout <<"D2= "<< c << "\n"; show\_B();}

void show\_D3() { cout << "D3= "<<d << "\n"; show\_D1(); show\_D2();}

void show\_D4() { cout <<"D4= "<< e << "\n"; show\_D1(); show\_D2();}

Повний текст програми

#include <iostream.h>

#include <stdlib.h>

class B{

int a;

public:

B() { };

B(int x) { a=x; }

void show\_B() { cout <<"B= "<< a << "\n"; }

};

class D1: public B {

int b;

public:

D1(int x, int y) : B(y) { b=x;};

void show\_D1() { cout <<"D1= "<< b << "\n"; show\_B();}

};

class D2: private B{

int c;

public:

D2(int x, int y) : B(y) { c=x;};

void show\_D2() { cout <<"D2= "<< c << "\n"; show\_B();}

};

class D3: private D1, public D2 {

int d;

public:

D3(int x, int y, int z, int i, int j) : D1(y,z), D2(i,j) { d=x;}

void show\_D3() { cout << "D3= "<<d << "\n"; show\_D1(); show\_D2();}

};

class D4: public D2, private D1 {

int e;

public:

D4(int x, int y, int z, int i, int j) : D1(y,z), D2(i,j) { e=x;}

void show\_D4() { cout <<"D4= "<< e << "\n"; show\_D1(); show\_D2();}

};

main() {

D3 temp(100,200,300,400,500);

D4 temp1(1,2,3,4,5);

cout << "D3 temp(100,200,300,400,500);\n";

cout << "D4 temp1(1,2,3,4,5);\n";

cout<< "\n Відповідно до ієрархії класу D3: \n";

temp.show\_D3();

cout<< "\n Відповідно до ієрархії класу D4\n";

temp1.show\_D4();

system(―pause‖);

return 0;

}