**Лекція 24. Множинне наслідування.**

**Визначення.**

Досі ми розглядали просте наслідування, коли похідний клас утворювався з одного базового. В С++ похідний клас можна утворювати з кількох базових класів. Множинне наслідування (multiple inheritance) застосовується в тому випадку, коли об'єкт являє собою поняття, що поєднує в собі кілька загальних понять, кожне з яких може бути представлено базовим класом. Можливість мати більше одного базового класу тягне за собою можливість неодноразового входження класу як базового. Така ситуація виникає, коли базові класи мають спільного предку.

Множинне наслідування від простого відрізняється лише тим, що похідний клас наслідує всі властивості базових.

Оголошення похідного класу при множинному наслідуванні:

**class ім*'*я*\_*похідного*\_*класу:**

**специфікатор*\_*доступу ім*'*я*\_*базового*\_*класу*1,***

***...,***

**специфікатор*\_*доступу ім*'*я*\_*базового*\_*классаN**

**{**

**// тіло класу**

**};**

Аналогічно, як i в простому наслідуванні, відсутність специфікатора доступу по замовчуванню інтерпретується закритим специфікатором (private).

Приклад:

**#include <stdio.h>**

**#include <Windows.h>**

**class TInner**

**{**

**public:**

**int i;**

**TInner(int n):i(n){printf("Ctor Inner\n");}**

**TInner(TInner& x){ \*this = x; printf("Copy ctor TInner\n");}**

**~TInner(){printf("Dtor TInner\n");}**

**};**

**class TBase1**

**{**

**public:**

**double a;**

**TInner b;**

**TBase1():b(10),a(5.0){printf("Ctor TBase1\n");}**

**TBase1(TBase1& x):b(10){ \*this = x; printf("Copy ctor TBase1");}**

**~TBase1(){printf("Dtor TBase1\n");}**

**void printBase() {printf("TBase1::TInner::i = %d a = %lf\n",b.i, a);}**

**};**

**class TBase2**

**{**

**public:**

**char c;**

**TBase2():c('Z'){printf("Ctor TBase2\n");}**

**TBase2(TBase2& x){\*this = x; printf("Copy ctor TBase2");}**

**~TBase2(){printf("Dtor TBase2\n");}**

**void printBase2() {printf("TBase2::c = c = %c\n",c);}**

**};**

**class TDerived: public TBase1, public TBase2**

**{**

**public:**

**float f;**

**TDerived():f(10.0){printf("Ctor TDerived\n");}**

**~TDerived(){printf("Dtor TDerived\n");}**

**void printDerived() {printf("TDerived::%lf\n",f);}**

**};**

**int main()**

**{ system("color F0");**

**TDerived obj;**

**printf("TBase1::a = %lf \n",obj.a);**

**printf("TBase1::b = %d \n",obj.b.i);**

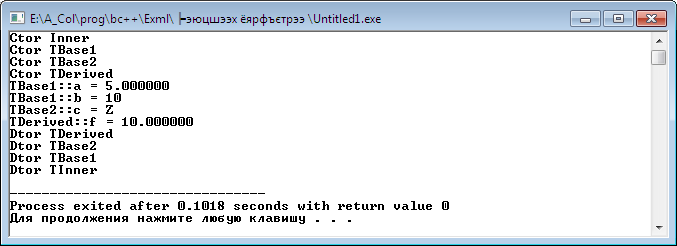
**printf("TBase2::c = %c \n",obj.c);**

**printf("TDerived::f = %lf \n",obj.f);**

**return 0;**

**}**

У класу TDerived2 є два базових класи, кожний з який є відкритим. Отже, об'єкт класу TDerived2 містить усі поля, що належать класам TBase1 і TBase2, причому усі вони відкриті. Програма виводить на екран наступні повідомлення.



Перші чотири рядки являють собою повідомлення від конструкторів у порядку створення об'єктів — внутрішній, два базових і похідний. Потім функція printf() виводить на екран поля об'єкта obj, що він успадкував від класів TBase1 і TBase2, а також його власне поле. Наприкінці програми викликаються деструктори, що знищують об'єкти в зворотному порядку.

**Конструктори і деструктори при множинному наслідуванні.**

При створенні об'єктів похідного класу необхідно спочатку створити об'єкт базового класу, оскільки об'єкт похідного класу являє собою модифікований об'єкт базового класу. Тому спочатку викликається конструктор базового класу, а потім — конструктор похідного класу. Якщо базових класів декілька, вони викликаються в порядку їх перерахування в списку наслідуваних класів. Деструктори викликаються в зворотному порядку.

Оголошення конструктора похідного класу.

**ім*'*я*\_*похідного*\_*класу:**

**ім*'*я*\_*базового класу*1*(параметри), ..., ім*'*я*\_*базового класу N (параметри)**

**{**

**тіло конструктора**

**}**

Наприклад, у приведеній вище програмі конструктор похідного класу визначається в такий спосіб.

**TDerived():TBase1(),TBase2(){f=15.0; printf("Ctor TDerived\n");}**

В класі, виведеному з кількох базових, може виникнути необхідність у виклику конструкторів цих класів. Якщо класи А, В, С мають конструктори по замовчуванню, в похідному класі D їх можна викликати наступним чином:

class D:public A,public B,public C{

public:

D( ):A( ),B( ),C( ){ }

//...};

Або ж конструктор можна оголосити так:

class D:public A,public B,public C{

public:

D( );

//...};

і реалізувати його за межами формального опису класу:

D::D( ):A( ),B( ),C( ){ }

Конструктори базових класів також можуть викликатися неявно, що у приведеній вище програмі виконується при виклику конструктору.

**TDerived():f(15.0) { printf("Ctor TDerived\n");}**

Конструктори базових класів у списку не можна перемежовувати списками ініціалізації. Наприклад, така конструкція непрацездатна.

TDerived():TBase1(),TBase2():f(15.0) {printf("Ctor TDerived\n");}

**Застосування конструкторів з параметрами**

У базових класах можуть бути конструктори з параметрами. Вони викликаються аналогiчно як i в звичайному наслiдуваннi пiсля двокрапки при визначеннi конструктора. Конструктори в похiдному класi викликаються в тому порядку, в якому оголошенi базовi класи. Навіть якщо в конструкторі похідного класу вказати послідовність виклику іншою (відмінною від порядку слідування базових класів), це не вплине на результат.

Модифікуємо попередній приклад (змінені рядки коду відмічені кольором).

**#include <stdio.h>**

**#include <Windows.h>**

**using namespace std;**

**class TInner**

**{**

**public:**

**int i;**

**TInner(int n):i(n){printf("Ctor Inner\n");}**

**TInner(TInner& x){ \*this = x; printf("Copy ctor TInner\n");}**

**~TInner(){printf("Dtor TInner\n");}**

**};**

**class TBase1**

**{**

**public:**

**double a;**

**TInner b;**

**TBase1(int x, double y):b(x),a(y){printf("Ctor TBase1\n");}**

**TBase1(TBase1& x):b(10){ \*this = x; printf("Copy ctor TBase1");} //**

**~TBase1(){printf("Dtor TBase1\n");}**

**void printBase() {printf("TBase1::TInner::i = %d a = %lf\n",b.i, a);}**

**};**

**class TBase2**

**{**

**public:**

**char c;**

**TBase2(char x):c(x){printf("Ctor TBase2\n");}**

**TBase2(TBase2& x){\*this = x; printf("Copy ctor TBase2");}**

**~TBase2(){printf("Dtor TBase2\n");}**

**void printBase2() {printf("TBase2::c = c = %c\n",c);}**

**};**

**class TDerived: public TBase1, public TBase2**

**{**

**public:**

**float f;**

**TDerived(float x):TBase1(20.0, 30),TBase2('X')**

**{ f = x;printf("Ctor TDerived\n"); }**

**~TDerived(){printf("Dtor TDerived\n");}**

**void printDerived2() {printf("TDerived2::%lf\n",f);}**

**};**

**int main()**

**{ system("color F0");**

**TDerived obj(15.0);**

**printf("TBase1::a = %lf \n",obj.a);**

**printf("TBase1::b = %d \n",obj.b.i);**

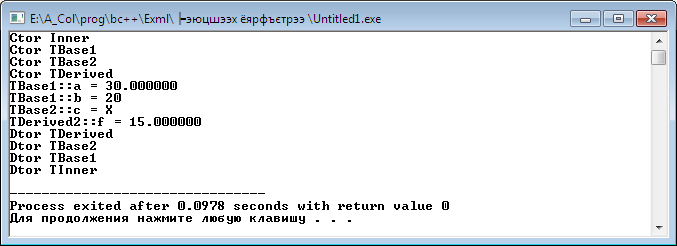
**printf("TBase2::c = %c \n",obj.c);**

**printf("TDerived2::f = %lf \n",obj.f);**

**return 0;**

**}**

Результат.



**Передача параметрів конструкторам базових класів**

Параметри конструктора базового класу можна передавати конструкторам похідних класів. Модифікуємо попередній приклад (змінені рядки коду відмічені кольором).

**#include <stdio.h>**

**class TInner**

**{**

**public:**

**int i;**

**TInner(int n):i(n){printf("Ctor Inner\n");}**

**TInner(TInner& x){ \*this = x; printf("Copy ctor TInner\n");}**

**~TInner(){printf("Dtor TInner\n");}**

**};**

**class TBase1**

**{**

**public:**

**double a;**

**TInner b;**

**TBase1(int x, double y):b(x),a(y){printf("Ctor TBase1\n");}**

**TBase1(TBase1& x):b(10){ \*this = x; printf("Copy ctor TBase1");}**

**~TBase1(){printf("Dtor TBase1\n");}**

**void printBase() {printf("TBase1::TInner::i = %d a = %lf\n",b.i, a);}**

**};**

**class TBase2**

**{**

**public:**

**char c;**

**TBase2(char x):c(x){printf("Ctor TBase2\n");}**

**TBase2(TBase2& x){\*this = x; printf("Copy ctor TBase2");}**

**~TBase2(){printf("Dtor TBase2\n");}**

**void printBase2() {printf("TBase2::c = c = %c\n",c);}**

**};**

**class TDerived: public TBase1, public TBase2**

**{**

**public:**

**float f;**

**TDerived(int x, double y, char z, float w):TBase1(x, y),TBase2('z')**

**{ f = w;printf("Ctor TDerived\n"); }**

**~TDerived(){printf("Dtor TDerived\n");}**

**void printDerived2() {printf("TDerived2::%lf\n",f);}**

**};**

**int main()**

**{**

**TDerived obj(1.0, 2, 'Y', 3.0);**

**printf("TBase1::a = %lf \n",obj.a);**

**printf("TBase1::b = %d \n",obj.b.i);**

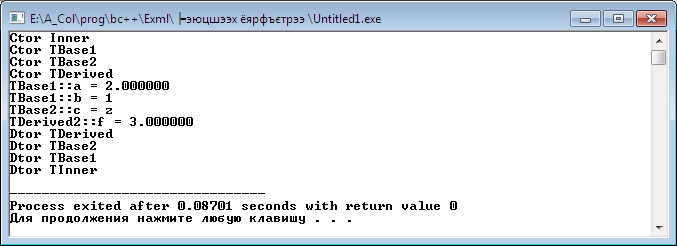
**printf("TBase2::c = %c \n",obj.c);**

**printf("TDerived2::f = %lf \n",obj.f);**

**return 0;**

**}**

Результат.



**Віртуальні базові класи - Діамантове успадкування**

Розглянемо приклад.

**#include <iostream>**

**#include <Windows.h>**

**using namespace std;**

**class PoweredDevice**

**{**

**public:**

**PoweredDevice(int power)**

**{**

**std::cout << "PoweredDevice: " << power << '\n';**

**}**

**};**

**class Scanner: public PoweredDevice**

**{**

**public:**

**Scanner(int scanner, int power)**

**: PoweredDevice(power)**

**{**

**std::cout << "Scanner: " << scanner << '\n';**

**}**

**};**

**class Printer: public PoweredDevice**

**{**

**public:**

**Printer(int printer, int power)**

**: PoweredDevice(power)**

**{**

**std::cout << "Printer: " << printer << '\n';**

**}**

**};**

**class Copier: public Scanner, public Printer**

**{**

**public:**

**Copier(int scanner, int printer, int power)**

**: Scanner(scanner, power), Printer(printer, power)**

**{**

**}**

**};**

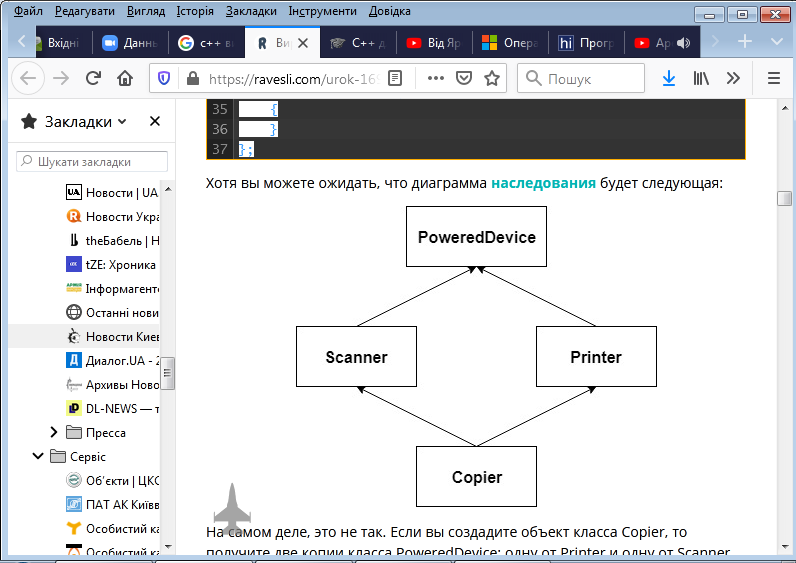
**int main()**

**{system("color F0");**

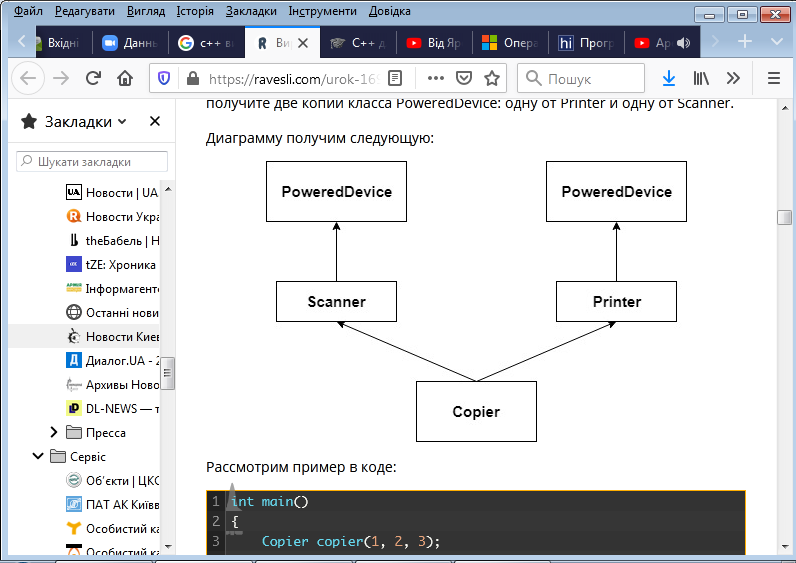
**Copier copier(1, 2, 3);**

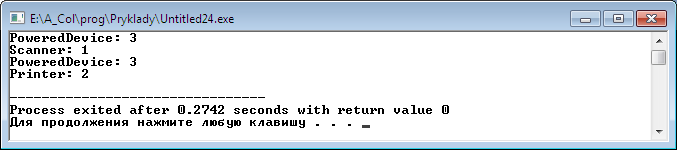
**}**

В цьому прикладі очікується така діаграма:



Насправді діаграма буде така:



****

Доступ до члена базового класу вважається неоднозначним, якщо вираз, що використовується для доступу, задає більше однієї функції, об'єкта, типу або елемента переліку. Перевірка на однозначність відбувається до перевірки можливості доступу. До специфікації базового класу можна додати службове слово **virtual**. **Віртуальний базовий клас** — це клас, об’єкт якого є спільним для використання усіма похідними класами.

Окремий об'єкт віртуального базового класу V розділяється між всіма класами, які вказали V, визначаючи свої базові класи, наприклад:

**class** A : **virtual** **public** V

{

. . .

};

**class** B : **virtual** **public** V

{

. . .

};

**class** C : **public** A, **public** B

{

. . .

};

Тут об'єкт класу C матиме тільки один вкладений об'єкт класу V (ромбовидне або діамантове наслідування). Конструктори віртуальних базових класів викликаються до будь-яких конструкторів невіртуальних базових класів.

Множинне успадкування часте приводить до неоднозначностей. Розглянемо класичний варіант, що одержав назву діамантового успадкування, — клас TBase є базовим для класів TDerived1 і TDerived2, а ті, у свою чергу, — базовими для класу TNext (див. Рис.1). Відповідно до механізму успадкування всі члени класу TBase стають членами класів TDerived1 і TDerived2, потім вони успадковуються класом TNext. Оскільки при першому успадкуванні (одиночному) члени класу TBase “роздвоюються”, при множинному успадкуванні в класі TNext виникає неоднозначність — як бути з членами класу TBase?

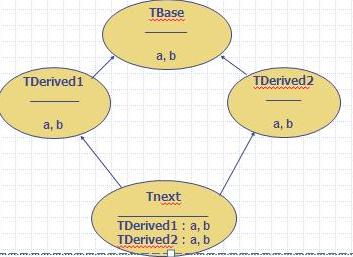


Рис.1. Приклад Діамантового успадкування

Розглянемо приклад.

**#include <stdio.h>**

**#include <Windows.h>**

**using namespace std;**

**class TInner**

**{**

**public:**

**int i;**

**TInner(int n):i(n){printf("Ctor Inner\n");}**

**TInner(TInner& x){ \*this = x; printf("Copy ctor TInner\n");}**

**~TInner(){printf("Dtor TInner\n");}**

**};**

**class TBase**

**{**

**public:**

**double a;**

**TInner b;**

**TBase(int x, double y):b(x),a(y){printf("Ctor TBase\n");}**

**TBase(TBase& x):b(10){ \*this = x; printf("Copy ctor TBase");}**

**~TBase(){printf("Dtor TBase\n");}**

**void printBase() {printf("TBase::TInner::i = %d a = %lf\n",b.i, a);}**

**};**

**class TDerived1: public TBase**

**{**

**public:**

**char c;**

**TDerived1(int x, int y, char z):TBase(x,y)**

**{c = z; printf("Ctor TDerived1\n");}**

**~TDerived1(){printf("Dtor TDerived1\n");}**

**void printDerived1() {printf("TDerived1::c = %c\n",c);}**

**};**

**class TDerived2: public TBase**

**{**

**public:**

**double d;**

**TDerived2(int x, int y, double z):TBase(x,y)**

**{d = x; printf("Ctor TDerived2\n");}**

**~TDerived2(){printf("Dtor TDerived2\n");}**

**void printDerived2() {printf("TDerived2::c = c = %lf\n",d);}**

**};**

**class TNext:public TDerived1, public TDerived2**

**{**

**public:**

**float f;**

**TNext(int x, int y, char z, double w):TDerived1(x,y,z),TDerived2(x,y,w)**

**{f = w;printf("Ctor TDerived\n");}**

**~TNext(){printf("Dtor TNext\n");}**

**void printNext() {printf("Tnext::%lf\n",f);}**

**};**

**int main()**

**{system("color F0");**

**TNext obj(1.0, 2, 'Y', 3.0);**

**// printf("TBase::a = %lf \n",obj.a); // Неоднозначність!**

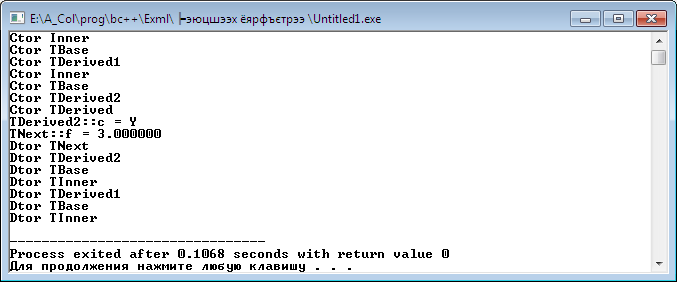
**// printf("TDerived1::b = %d \n",obj.b.i); // Неоднозначність!**

**printf("TDerived2::c = %c \n",obj.c);**

**printf("TNext::f = %lf \n",obj.f);**

**return 0;**

**}**



Очевидно, що поля a і b попадають із класу TBase у клас TNext двома шляхами — через клас TDerived1 і TDerived2. Отже, звертання obj.a і obj.b.i є неоднозначними(в коді ці оператори закоментовані, оскільки викликають помилку компіляції).

**// printf("TBase::a = %lf \n",obj.a); // Неоднозначність!**

**// printf("TDerived1::b = %d \n",obj.b.i); // Неоднозначність!**

Вирішити проблему можна двома шляхами. Наприклад, щоб уточнити ім'я поля можна використовувати операцію розширення області видимості для визначення необхідного члена класу:

**printf("TBase::a = %lf \n",obj.TDerived1::a);**

**printf("TDerived1::b = %d \n",obj.TDerived1::b.i);**

Однак залишається проблема, зв'язана з розміром класу. Хоча неоднозначність усунута, клас TNext як і раніше містить подвоєний набір змінних, успадкованих від класу TBase. Існує більш елегантний спосіб рішення цієї проблеми — віртуальні базові класи. Для того щоб запобігти дублювання, у списку успадкування перед ім'ям базового класу варто вказати ключовому слову virtual. Правда, тепер необхідно самому передбачити конструктор за замовчуванням.

**#include <stdio.h>**

**#include <Windows.h>**

**using namespace std;**

**class TInner**

**{**

**public:**

**int i;**

**TInner(int n):i(n){printf("Ctor Inner\n");}**

**TInner(TInner& x){ \*this = x; printf("Copy ctor TInner\n");}**

**~TInner(){printf("Dtor TInner\n");}**

**};**

**class TBase**

**{**

**public:**

**double a;**

**TInner b;**

**TBase():a(0),b(0){}**

**TBase(int x, double y):b(x),a(y){printf("Ctor TBase\n");}**

**TBase(TBase& x):b(10){ \*this = x; printf("Copy ctor TBase");}**

**~TBase(){printf("Dtor TBase\n");}**

**void printBase() {printf("TBase::TInner::i = %d a = %lf\n",b.i, a);}**

**};**

**class TDerived1: virtual public TBase**

**{**

**public:**

**char c;**

**TDerived1(int x, int y, char z):TBase(x,y){c = z;**

**printf("Ctor TDerived1\n");}**

**~TDerived1(){printf("Dtor TDerived1\n");}**

**void printDerived1() {printf("TDerived1::c = %c\n",c);}**

**};**

**class TDerived2: virtual public TBase**

**{**

**public:**

**double d;**

**TDerived2(int x, int y, double z):TBase(x,y)**

**{ d = x; printf("Ctor TDerived2\n"); }**

**~TDerived2(){printf("Dtor TDerived2\n");}**

**void printDerived2() {printf("TDerived2::c = c = %lf\n",d);}**

**};**

**class TNext:public TDerived1, public TDerived2**

**{**

**public:**

**float f;**

**TNext(int x, int y, char z, double w):TDerived1(x,y,z),TDerived2(x,y,w)**

**{f = w;printf("Ctor TNext\n");}**

**//TDerived(float x):f(x) { printf("Ctor TDerived\n");}**

**~TNext(){printf("Dtor TNext\n");}**

**void printNext() {printf("TNext::%lf\n",f);}**

**};**

**int main()**

**{system("color F0");**

**TNext obj(1.0, 2, 'Y', 3.0);**

**printf("TBase::a = %lf \n",obj.a);**

**printf("TDerived1::b = %d \n",obj.b.i);**

**printf("TDerived2::c = %c \n",obj.c);**

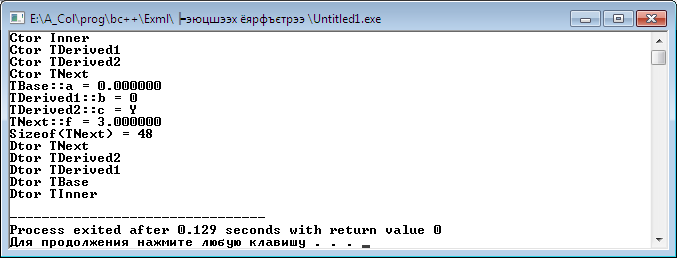
**printf("TNext::f = %lf \n",obj.f);**

**printf("Sizeof(TNext) = %d\n",sizeof(TNext));**

**return 0;**

**}**

Результати роботи цієї програми приведені нижче.



Як бачимо, механізм віртуальних базових класів дозволяє не тільки усунути неоднозначність, але і взагалі запобігти дублювання наслідуваних змінних — розмір класу TNext тепер дорівнює не 56 байт, а лише 48.

**Приклад**. Пегас — чарівний крилатий кінь.

Новий клас можна отримати від декількох базових класів - класи Кінь та класу Птах. Обидва класи Кінь та Птах відносяться до класу Тварини Кожний раз при створенні об’єкту класу Пегас створюється частина класу Кінь та частина класу Птах. При видаленні об’єкту класу Пегас видаляються частини, що відносяться до класу Кінь та класу Птах.

Для цього клас Тварини треба оголосити як віртуальний базовий клас для похідних класів Кінь та Птах. Клас Тварин при цьому ніяк не змінюється. В класах Кінь та Птах при оголошенні вказується віртуальність наслідування від базового класу Тварин. А от клас Пегас змінюється суттєво. Переважно конструктори класу ініціалізують власні змінні та змінні-члени базового класу. При віртуальному наслідуванні змінні основного базового класу ініціалізуються конструктором класу, який є останнім в ієрархії класів. Отже клас Тварин буде ініціалізований конструктором класу Пегас.

 class Animal

{ …..

}

class Horse : virtual public Animal

{ …

}

class Bird : virtual public Animal

{ …

}

class Pegasus : public Horse, public Bird

{ …

}

*Для самостійного вивчення*: Поглибити матеріал лекції за наданою літературою. Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи, запропонованих на лекції.

*Література*

1. Джейс Либерти Освой самостоятельно С++ за 21 день: 3-е изд. пер. с англ.: Уч. пос. – М.: Издательский дом „Вильямс”, 2001. – 816 с.: ил..
2. Павловская Т.А. С/С++. Программирование на языке высокого уровня. СПб.: Питер, 2003. – 461 с. URL:  <http://www.ph4s.ru/bookprogramir_1.html>
3. Липпман С. Б., Лажойе Ж. Язык программирования С++: Вводный курс. — М.: ДМК, 2001. URL: <http://www.insycom.ru/html/metodmat/inf/Lipman.pdf>
4. Дейтел Х., Дейтел П. Основы программирования на С++. – М.: Бином, 1999. – 1024 с. URL:  <http://ijevanlib.ysu.am/wp-content/uploads/2018/03/deytel.pdf>
5. Бадд Т. Объектно-ориентированное программирование в действии. [2-е изд.] – СПб.: Изд-во "Питер". 1997.  URL: <http://khizha.dp.ua/library/Timothy_Budd_-_Introduction_to_OOP_(ru).pdf>.
6. *Герб Саттер,Андрей Александреску* "Стандарты программирования на С++ ", [*Вильямс, 2005*](http://www.williamspublishing.com/)*; 304 с.*
7. *Скотт Мейерс.* Эффективное использование C++. 50 рекомендаций по улучшению ваших программ и проектов. *"ДМК", 2000; 240 с.*
8. *Скотт Мейерс.* Наиболее эффективное использование C++. 35 новых рекомендаций по улучшению ваших программ и проектов.*"ДМК",2000;304 с.*
9. Г.Буч. Об‘єктно-орієнтоване проектування з прикладами застосування. – К.: Видавничий центр "Академія". 2002. ­ 499 с.
10. Страуструп Б. Язык программирования С++· Киев: "ДиаСофт", 1993. - 256 с.

*Контрольні запитання для самоперевірки*.

1. Дайте визначення множинному наслідуванню.
2. Як реалізується множинне наслідування ?
3. Які небажані ситуації виникають при множинному наслідуванні ?
4. Як здійснюється передача параметрів конструкторам базових класів при множинному наслідуванні?
5. Як застосовуються конструктори з параметрами при множинному наслідуванні?

***Контрольні запитання для надання письмових відповідей***.

З погляду створення Вами демонстраційної програми для ілюстрації діамантового наслідування:

1. В прикладі Пегас які об’єкти-члени класу Ви можете визначити для класів Тварини, Кінь, Птах, Пегас?
2. В прикладі Пегас які методи класу Ви можете визначити для класів Тварини, Кінь, Птах, Пегас?