ЛАБОРАТОРНА РОБОТА №7
ВІРТУАЛЬНІ ФУНКЦІЇ І ПОЛІМОРФІЗМ

Мета роботи: отримати навички створення ієрархії класів з використанням простого наслідування і абстрактного класу. Вивчення поліморфізму і віртуальних функцій.

Основні теоретичні відомості

Одним з трьох основних принципів об'єктно-орієнтованого програмування є поліморфізм. Стосовно до C++ поліморфізм є термін, який використовується для опису процесу, в якому різні реалізації функції можуть бути доступні за допомогою одного і того ж імені. З цієї причини поліморфізм іноді характеризується фразою «один інтерфейс, багато методів». Це означає, що до всіх функцій-членів загального класу можна отримати доступ одним і тим же способом, незважаючи на можливі відмінності в конкретних діях, пов'язаних з кожною окремою операцією.

У C++ поліморфізм підтримується як під час виконання, так в період компіляції програми. Перевантаження операторів і функцій – це приклади поліморфізму, що відноситься до часу компіляції. Але, не дивлячись на могутність механізму перевантаження операторів і функцій, він не в змозі вирішити всі завдання, які виникають в реальних додатках об'єктно-орієнтованої мови програмування. Тому в C++ також реалізований поліморфізм періоду виконання на основі використання похідних класів і віртуальних функцій.

Вказівники на похідні типи

Фундаментом для динамічного поліморфізму служить вказівник на базовий клас. Вказівники на базові і похідні класи пов'язані такими відносинами, які не властиві вказівникам інших типів. Вказівник одного типу, як правило, не може вказувати на об'єкт іншого типу. Однак вказівники на базові класи і об'єкти похідних класів – виключення з цього правила. У C++ вказівник на базовий клас також можна використовувати для посилання на об'єкт будь-якого класу, виведеного з базового. Наприклад, припустимо, що у

нас є базовий клас Animal і класи Cat, God, Frog і Cow, які виведені з класу Animal. (Рис. 7.1.) У C++ вказівник, що був оголошений як вказівник на базовий клас Animal, може бути також вказівником на похідні класи Cat, God, Frog і Cow.

```
Animal *A[4]; // вказівники на базовий клас
Cat B; Dog C; Frog D; Cow E;
A[0] = &B; // вказівник на базовий клас посилається на
//об'єкт класу Cat
A[1] = &C; // вказівник на базовий клас посилається на
//об'єкт класу Dog
A[2] = &D; // вказівник на базовий клас посилається на
//об'єкт класу Frog
A[3] = &E; // вказівник на базовий клас посилається на
//об'єкт класу Cow
```

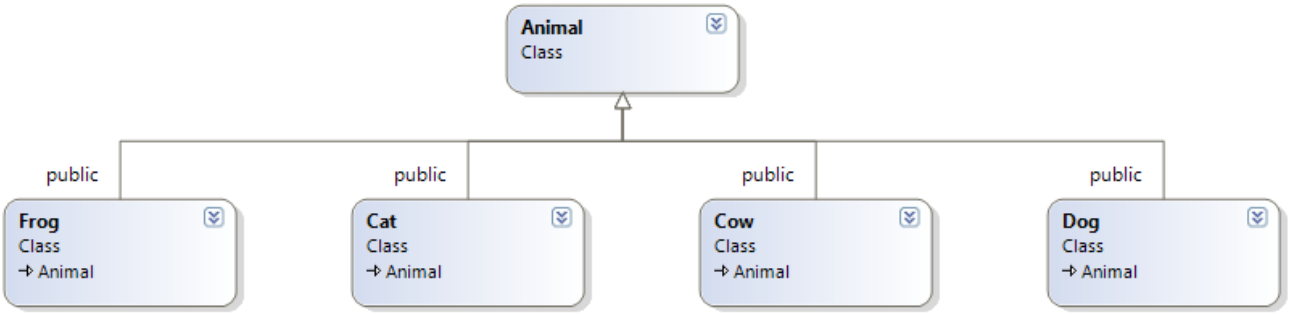


Рис. 7.1. Класи Cat, God, Frog і Cow виведени з класу Animal.

Крім того, необхідно розуміти, що хоча «базовий» вказівник можна використовувати для доступу до об'єктів будь-якого похідного типу, зворотне твердження не вірно.

Той факт, що вказівник на базовий тип можна використовувати для посилання на будь-який об'єкт, виведений з базового, надзвичайно важливий і принциповий для C++. Ця гнучкість є ключовим моментом для способу реалізації динамічного поліморфізму в C++.

Подібно вказівниками, *посилання* на базовий клас також можна використовувати для доступу до об'єкта похідного типу. Ця можливість особливо часто застосовується при передачі аргументів функцій. Параметр, який має тип посилання на базовий клас, може приймати об'єкти базового класу, а також об'єкти будь-якого іншого типу, виведеного з нього.

Віртуальні функції

Динамічний поліморфізм можливий завдяки поєднанню двох засобів: спадкування і віртуальних функцій.

Віртуальна функція – це функція, яка оголошується в базовому класі з використанням ключового слова virtual і перевизначається в одному або декількох похідних класах. Таким чином, кожен похідний клас може мати власну версію віртуальної функції.

Коли віртуальна функція викликається через вказівник (або посилання) на базовий клас, C++ визначає, яку саме версію віртуальної функції потрібно викликати, по типу об'єкта, що адресується цим вказівником. Причому слід мати на увазі, що це рішення приймається під час виконання програми. Отже, при посиланні на різні об'єкти будуть викликатися і різні версії віртуальної функції. Саме за типом об'єкта, що адресується, (а не за типом самого вказівника) визначається, яка версія віртуальної функції буде виконана. Таким чином, якщо базовий клас містить віртуальну функцію і якщо з цього базового класу виведено два (або більше) інших класів, то при адресації різних типів об'єктів через вказівник на базовий клас будуть виконуватися і різні версії віртуальної функції. Аналогічний механізм працює і при використанні посилання на базовий клас.

Щоб оголосити функцію віртуальної, досить випередити її оголошення ключовим словом virtual.

Функція оголошується віртуальною в базовому класі за допомогою ключового слова virtual. При перевизначенні віртуальної функції в похідному класі ключове слово virtual повторювати не потрібно (хоча це не буде помилкою).

Атрибут virtual передається «у спадок».

Якщо функція оголошується як віртуальна, вона залишається такою незалежно від того, через скільки рівнів похідних класів вона може пройти.

Якщо віртуальна функція перевизначається в похідному класі, її називають перевизначенною.

Прототипи віртуальної функції і її перевизначень повинні бути абсолютно однаковими. Якщо прототипи будуть різними, то така функція буде просто вважатися перевантаженою, і її «віртуальна сутність» втратиться.

І ще одне важливе зауваження. Конструктор не успадковується. Конструктор базового класу створює об’єкт, який «живе» в об'єкті похідного класу. Тому деструкторам дозволяється бути віртуальними, а конструкторам - ні.

```
class Cat : public
{
public:
    virtual ~Cat()
    {
    }
```

```
class Cat : public Animal
{
public:
    virtual Cat()
    {
    }
```

Виші показані два фрагмента коду, зліва правильний, а справа - з помилкою.

Клас, який включає віртуальну функцію, називається поліморфним класом. Цей термін також застосовується до класу, який успадковує базовий клас, що містить віртуальну функцію.

Розглянемо наступну коротку програму, в якій демонструється використання віртуальних функцій.

```
class Animal
{
public:
 virtual void GetSound(){
 cout<<"Тварина видає просто якийсь звук "<<endl;
 }
};

class Cat : public Animal
{
public:


 void GetSound() // Перевизначення функції GetSound()
                 //для класу Cat
 {
      cout<<"Мяу..."<<endl;
 }
```

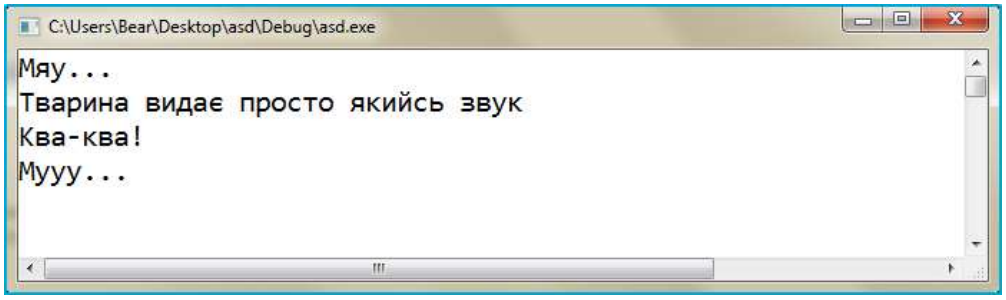
```
};
class Dog : public Animal
{
public:
 /*void GetSound() // Перевизначення функції GetSound()
                    //для класу Dog закоментоване
 {
      cout<<"Гав!"<<endl;
 }*/
};
class Frog : public Animal
{
public:
 void GetSound() // Перевизначення функції GetSound()
                  //для класу Frog
 {
      cout<<"Ква-ква!"<<endl;
 }
};

class Cow : public Animal
{
public:
 void GetSound() // Перевизначення функції GetSound()
                         //для класу Cow
 {
      cout<<"Муууу..."<<endl;
 }
};

int main()
{
Animal *A[4]; // вказівники на базовий клас
Cat B; Dog C; Frog D; Cow E;
A[0] = &B; // вказівник на базовий плас посилається на
//об'єкт класу Cat
A[1] = &C; // вказівник на базовий плас посилається на
//об'єкт класу Dog
A[2] = &D; // вказівник на базовий плас посилається на
//об'єкт класу Frog
A[3] = &E; // вказівник на базовий плас посилається на
//об'єкт класу Cow
for(int i=0; i<4; i++)
      A[i]->GetSound(); // поліморфний виклик!!!

}
```

При виконанні програма генерує ось такі результати.



Якщо похідний клас не перевизначає віртуальну функцію, то використовується функція, яка визначена в базовому класі.

Як бачимо, в похідному класі Dog функція `GetSound()` не перевизначається и тоді викликається функція базового класу.

Поліморфізм забезпечує можливість деякому узагальненому класу визначати функції, які будуть використовувати всі похідні від нього класи, причому похідний клас може визначити власну реалізацію деяких або всіх цих функцій.

Таким чином можна визначити поліморфізм як один інтерфейс для безлічі реалізацій.

Чисто віртуальні функції та абстрактні класи

У багатьох випадках взагалі немає сенсу давати визначення віртуальної функції в базовому класі. Наприклад, в базовому класі Animal (з попереднього прикладу) визначення функції GetSound() – це просто вивід тексту без особливого сенсу, тому, що неможливо собі уявити, який звук видає просто тварина.

Конкретний звук завжди видає конкретна тварина, тому функцію GetSound() обов'язково треба перевизначити во всіх похідних класах.

Чисто віртуальна функція – це віртуальна функція, яка не має визначення в базовому класі.

Чисто віртуальна функція – це функція, яка оголошена в базовому класі, але не має в ньому ніякого визначення. Тому будь-який похідний тип повинен визначити власну версію цієї функції, адже у нього просто немає ніякої можливості використовувати версію з базового класу (через її відсутність).

Щоб оголосити чисто віртуальну функцію, використовується наступний загальний формат:

```
virtual тип імя_функциі (спісок_параметров) = 0;
```

Тут під елементом тип мається на увазі тип значення, що повертається функцією, а елемент імя_функції – її ім'я. Позначення = 0 є ознакою того, що функція тут оголошується як чисто віртуальна.

Оголосивши функцію чисто віртуальної, програміст створює умови, при яких похідний клас просто змушений мати визначення власної її реалізації. Без цього компілятор видасть повідомлення про помилку.

```
class Animal
{
public:
 virtual void GetSound()=0; // Чисто віртуальна функція
};
class Cat : public Animal
{
public:
 void GetSound() // Перевизначення  GetSound() для класу Cat
 {
      cout<<"Мяу..."<<endl;
 }
};
class Dog : public Animal
{
public:
 /*void GetSound() // Перевизначення функції GetSound()
                   //для класу Cat закоментоване
 {
      cout<<"Гав!"<<endl;
 }*/
};
```

Наприклад, при компіляції програми, фрагмент якої представлений вище, з'являється ось таке вікно повідомлень (Рис. 7.2):

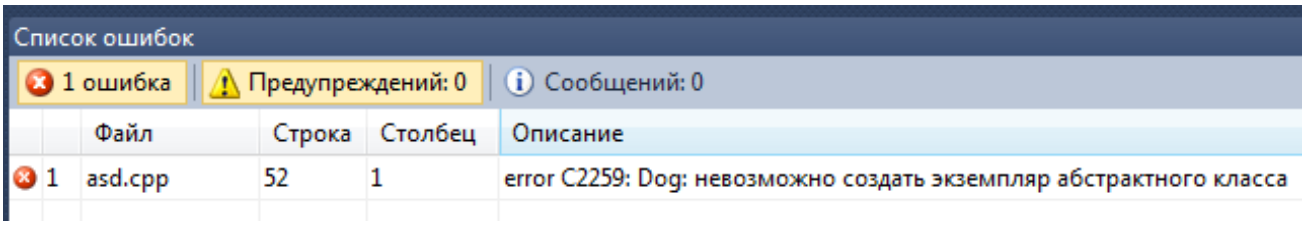


Рис. 7.2. Повідомлення під час компіляції програми.

Якщо клас має хоча б одну чисто віртуальну функцію, його називають абстрактним. Абстрактний клас характеризується однією важливою особливістю: у такого класу не може бути об'єктів. Абстрактний клас можна використовувати тільки в якості базового, з якого будуть виводитися інші класи. Причина того, що абстрактний клас не можна використовувати для створення об'єктів, лежить, безумовно, в тому, що його одна або кілька функцій не мають визначення. Але навіть якщо базовий клас є абстрактним, його все одно можна використовувати для оголошення вказівників і посилань, які необхідні для підтримки динамічного поліморфізму.

Завдання до лабораторної роботи

1. Визначити ієрархію класів (відповідно до варіанта). З переліку класів вибрати один, який буде стояти на чолі ієрархії. Переважно це має бути абстрактний клас.
2. Реалізувати класи. Створити поліморфну функцію (віртуальну або чисто віртуальну) відповідно варіанту.
3. Написати демонстраційну програму, в якій створюються об'єкти різних класів і поміщаються в масив або в список, після чого масив або список проглядається через виклик поліморфної функції.
4. Якщо базовий клас не абстрактний, зробити відповідну поліморфну функцію не віртуальною і подивитися, що буде. Пояснити отриманий результат.

Варіанти завдань:

1. Трикутник, геометрічна фігура, трапеція, рівносторонній трикутник. Поліморфна функція повертає значення площі відповідної фігури.
2. Куля, усічений прямий конус, геометрічне тіло, конус. Поліморфна функція повертає значення об’єму відповідного геометрічного тіла. (Формули для обчислення знайти самостійно)
3. Куля, геометрічне тіло, паралелепіпед, тетраедр. Поліморфна функція повертає значення площі поверхні тіла.

4. Кінь, слон, тура, ферзь, шахова фігура. Поліморфна функція приймає два параметра: перший-позиція фігури на шахівниці (наприклад, g2), а другий параметр – поле, куди треба піти (наприклад, h3), і виводить повідомлення – чи можливий такий хід, чи ні;

5. Кінь, слон, тура, ферзь, шахова фігура. Поліморфна функція приймає два параметра: перший – поле, де стоїть король противника (наприклад, g2), а другий параметр – позиція фігури на шахівниці (наприклад, h3), і виводить повідомлення – чи оголошений королю шах, чи ні;

6. Організація, страхова компанія, будівна компанія, метробуд. Поліморфна функція для страхової компанії виводить прізвище страхового агента, кількість полісів і загальну суму, для будівної компанії - прізвище прораба, загальну смету (суму в грн) і метраж (загальну площу кв.м), для метробуд – назва маршруту, кількість пасажирів, відстань в км.;

7. Ціле беззнакове число, двійкове число, десяткове число, шістнадцяткове число. Поліморфна функція виводить на екран число відповідно як двійкове, десяткове або шістнадцяткове.

8. Базовий клас – трикутник (поля – довжини двох сторін і кут між ними), похідні – рівносторонній, рівнобедрений, прямокутний. Поліморфна функція виводить на екран значення площі та периметра.

9. Двигун, вічний двигун (perpetuum mobile), двигун внутрішнього згоряння, електродвигун. Поліморфна функція для двигуна внутрішнього згоряння виводить робочий об'єм двигуна (л), потужність (л.с) і витрата бензину на 100 км., для електродвигуна - потужність (квт), напруга (в) і кількість фаз, ну а для perpetuum mobile - придумайте самі, щоб показати, що такий не існує.

10. Опір, активний опір, ємнісний опір, індуктивний опір. Поліморфна функція повертає значення опіру для відповідного типу і виводить повідомлення про тип опіру. Ланцюг змінного струму включає в себе певну кількість активних, ємнісних і індуктивних елементів.

активний опір $R = \frac{\rho l}{S}$; де р - питомий опір провідника, l - довжина провідника, S - площа перерізу провідника;

ємнісний опір $X_c = \frac{1}{\omega C}$; де ω -циклічна частота змінного струму (рад/с), C – ємність;

індуктивний опір $X_L = \omega L$, де L - індуктивність.

Контрольні запитання

1. Наведіть приклади поліморфізму з повсякденного життя.
2. Чи може конструктор бути віртуальним?
3. Чи може деструктор бути віртуальним?
4. Який клас називається абстрактним?
5. Чи успадковується віртуальність?
6. Коли клас називають поліморфним?
7. Чи можна створити об’єкт (екземпляр) абстрактного класу?
8. Чи можна створити вказівник на об’єкт (екземпляр) абстрактного класу?
9. Дайте визначення чисто віртуальної функції.