**Тема 13.** **Поліморфізм та віртуальні функції.** (4 години)

**Лекція 2.**

**Поліморфізм – це існування різних реалізацій однієї операції для об’єктів різного типу. Поліморфізм в мові програмування означає багатозначність змінних і функцій.**

Існує погляд, згідно з яким поліморфними вважаються операції, реалізовані лише віртуальними функціями (динамічний поліморфізм, або поліморфізм часу виконання програми). Клас, що містить хоча б один віртуальний метод, називають поліморфним класом. Головне, що поліморфні класи, завдяки динамічному зв'язуванню, надають можливість обробляти об'єкти, тип яких стає відомим лише під час виконання програми.

Проте частіше поліморфізм розуміють не лише як динамічний. Зокрема, перевантаження /перевизначення функцій та операторів у мові С++ реалізує статичний поліморфізм (поліморфізм часу компіляції).

Поліморфізм втілює принцип "один інтерфейс, кілька реалізацій". Він дозволяє використовувати спільний інтерфейс різних класів і не замислюватися над відмінностями в реалізації операцій, прихованих у класах. Це сприяє створенню ясного, короткого й гнучкого коду.

Приклад перевантаження функцій у класі.

Задано клас Day, що реалізує день тижня. У класі перевантажено функцію Set(), що встановлює новий день тижня. Метод Set() має 2 реалізації:

* без параметрів;
* з одним параметром.

Програмний код, що демонструє застосування “перевантаження” функцій у класі має наступний вигляд:

**#include <iostream>**

**#include <Windows.h>**

**using namespace std;**

**class Day**

**{**

**int day;**

**public:**

**// метод Set() перевантажено**

**void Set(int nday) { day = nday; }**

**void Set(void) { day = 1; }**

**int Get(void) { return day; }**

**};**

**int main()**

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

**Day D;**

**int day;**

**// використання перевантаженого метода Set**

**D.Set();**

**day = D.Get(); // day = 1**

**D.Set(5);**

**day = D.Get(); // day = 5**

**cout << day << endl;**

**return 0;**

**}**

##### Доступ до перевантаженої функції з допомогою покажчика на функцію

При оголошенні покажчика на “перевантажену” функцію, компілятор визначає потрібну функцію для покажчика за його сигнатурою при оголошенні.

**Наприклад.** Нехай задано 3 “перевантажені” функції Increment() для типів int, double, char. При оголошенні покажчика на функцію, потрібно явно задати тип покажчика при оголошенні. Нижченаведений код демонструє використання покажчика на перевантажену функцію

**#include <iostream>**

**#include <Windows.h>**

**using namespace std;**

**int Increment(int i)**

**{ return i+1; }**

**double Increment(double d)**

**{ return d+1; }**

**char Increment(char c)**

**{ return c+1; }**

**int main()**

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

**int (\*pi)(int); // покажчик на функцію, яка отримує**

**//параметром int і повертає int**

**pi = Increment; // p вказує на int Increment(int)**

**int d = (\*pi)(4); // d = 5**

**char (\*pc)(char);**

**pc = Increment; // p вказує на char Increment(char)**

**char c = (\*pc)('F'); // c = 'G'**

**cout << c << endl;**

**return 0;**

**}**

Для похідного класу можна визначати методи (функції), що мають такі ж імена, як і у методів базового класу. В цьому випадку має місце перевантаження функцій. Таким чином, похідний клас може не тільки доповнювати, але і коректувати поведінку базового класу. Доступ до перевизначеного (перевантаженого) методу базового класу для похідного класу виконується через ім'я, уточнене за допомогою операції доступу до області видимості (::).

Наприклад:

**class А**

**{**

**. . .**

**public:**

**void draw();**

**}**

**class В: public А**

**{**

**. . .**

**void draw();// перевизначення методу draw() базового класу**

**};**

**void draw();**

**{**

**class А:: draw(); //виклик функції draw() базового класу А**

**. . .**

**}**

Робота з об'єктами частіше проводиться через покажчики. Покажчику на базовий клас можна присвоїти значення адреси об'єкта будь-якого похідного класу. Якщо при використанні віртуальної функції виклик здійснюється за допомогою покажчика базового класу (або посилання), то С++ вибирає правильну перевизначену функцію у відповідному похідному класі, пов'язаному з даним об'єктом. Якщо функція-елемент визначена у базовому класі не як віртуальна, але перевизначена в похідному класі, і викликається через покажчик базового класу, то використовується функція базового класу. Якщо ж ця функція-елемент викликається через покажчик похідного класу, то використовується функція похідного класу. Це не поліморфне поводження.

Приклад виклику функції draw() через покажчик на базовий клас:

В b; // оголошення об’єкта похідного класу з ім’ям b

А \*pА; // оголошення покажчика pА на базовий клас А

pА =& b; // покажчику привласнюється адреса об'єкта b

pА-> draw (); // виклик функції draw() через покажчик на базовий клас

Отже, завдяки використанню віртуальних функцій і поліморфізму, один і той же виклик функції може привести до різних дій залежно від типу об'єкта, що приймає цей виклик.

**Зв’язки похідних класів з базовим**

Похідні класи мають з базовим класом зв'язки двох видів. **Перший** з них полягає в тому, що екземпляри похідних класів використовують всі відкриті члени базового класу – зокрема методи базового класу. Приклад.

**class Base**

**{ public:**

**void meth ()**

**{ cout << "In Base: meth() " << endl; }**

**};**

**class SubBase : public Base**

**{ // …**

**};**

**int main (void)**

**{**

**Base b; // екземпляр базового класу**

**SubBase sb; // екземпляр похідного класу**

**b.meth (); // виклик методу базового класу**

**sb.meth (); // виклик методу базового класу**

**return 0;**

**}**

Другий вид зв'язку полягає в тому, що:

* екземпляр базового класу можна створити як екземпляр похідного;
* посилання на базовий клас може посилатись на похідний;
* вказівник на базовий клас може вказувати на похідний.

Всі ці операції виконуються без явного приведення типів і є реалізацією відношення «is-a». Приклад.

**int main (void)**

**{**

**Base b;**

**SubBase sb;**

**// екземпляр базового класу створюється як похідний**

**Base bb = SubBase ();**

**// екземпляру базового класу присвоюється похідний**

**b = sb;**

**// посилання на базовий клас посилається на похідний**

**Base & bbb = sb;**

**// вказівник на базовий клас вказує на похідний**

**Base \*p = &sb;**

**// sb = b; // таке присвоєння неможливе!**

**return 0;**

**}**

Цілком зрозуміла заборона присвоєнь у зворотному напрямку – адже якщо екземпляр похідного класу створюється як базовий, то виникає проблема із викликом методів похідного класу, яких немає у базовому:

**class Base**

**{ public:**

**void meth ()**

**{ cout << "In Base: meth()" << endl; }**

**};**

**class SubBase : public Base**

**{**

**void meth\_SubBase ()**

**{ cout << "Власний метод похідного класу" << endl;}**

**};**

**int main (void)**

**{**

**Base b; // екземпляр базового класу**

**SubBase sb = b; // припустимо це можливим**

**b.meth (); // виклик методу базового класу**

**sb.meth\_SubBase (); // в базовому класі метод відсутній**

**return 0;**

**}**

Та обставина, що посилання та вказівники базового класу можуть вказувати на екземпляри похідних класів, приводить до такої можливості: методи, які мають параметрами посилання або вказівник на базовий клас, можуть викликатись із аргументами-екземплярами похідних класів:

**void fun (Base & b)**

**{**

**b.meth();**

**}**

**int main (void)**

**{**

**Base b;**

**SubBase sb;**

**fun (b); // так можливо**

**fun (sb); // і так теж можливо**

**return 0;**

**}**

Але в будь-якому разі, функція **fun()** викликатиме метод **meth()** базового класу.

Проте, можлива ситуація, коли успадковані методи похідних класів повинні поводити себе інакше, ніж методи базового класу. Така поведінка називається “поліморфною”. (Поліморфний – такий, що має багато форм). Реалізація поліморфного спадкування здійснюється одним із двох способів:

1) ***Використання віртуальних методів;***

***2)*** ***Перекриття методів базового класу у похідному класі.***

Використання віртуальних функцій ми розглядали в попередній лекції.

**Перекриття методів базового класу у похідному класі**

Розглянемо приклад.

**class Base**

**{ public:**

**void meth ()**

**{ cout << "In Base: meth() " << endl; }**

**};**

**class SubBase : public Base**

**{**

**// цей метод перекриває відповідний метод базового класу**

**void meth ()**

**{ cout << "In SubBase: meth() " << endl;}**

**};**

**int main (void) {**

**Base b; // екземпляр базового класу**

**SubBase sb; // екземпляр похідного класу**

**b.meth (); // виклик методу базового класу**

**sb.meth (); // виклик методу похідного класу**

**return 0;**

**}**

В усіх попередніх прикладах зв'язування екземпляру із конкретним методом (функцією-членом класу) відбувалось на етапі компіляції (тобто ще до початку її виконання). Ця процедура, як відомо, називається **раннім зв'язуванням**. Альтернативний спосіб – **пізнє** **зв'язування** (інколи – динамічне зв'язування, в С# - динамічний поліморфізм) дозволяє асоціювати об'єкт із методом саме **під час виконання програми**.

Якщо в похідному класі визначається метод, одноіменний з віртуальним методом базового класу, але з відмінною сигнатурою, він перекриває віртуальний метод базового класу. Це означає, що в похідних класах віртуальний метод базового класу не доступний. Приклад.

**class Base**

**{ public: // раніше визначені члени класу**

**virtual virt (); // віртуальний метод**

**};**

**class SubBase : public Base**

**{ public : // раніше визначені члени класу**

**virt (int i); // метод - перекриває віртуальний**

**};**

**int main (void)**

**{**

**Base b = Base ();**

**SubBase sb = SubBase ();**

**sb.virt (10); // припустимо**

**sb.virt (); //помилка – метод базового класу недоступний**

**return 0;**

**}**

Отже, віртуальна функція – це функція-член класу, помічена словом **virtual**, для якої можливе перевизначення у всіх або деяких похідних класах. При звертанні до екземпляру похідного класу ***через вказівник або посилання*** на базовий клас буде виконана саме перевизначена (заміщена) у похідному класі віртуальна функція.

**Поліморфна функція та змінна**

**Поліморфною** називається така змінна, яка може зберігати в собі значення різних типів даних

**Поліморфною функцією** є така функція, яка може бути викликана з аргументами різного типу, а фактичний код, що виконується залежить від типів аргументів.

**Поліморфний метод –** той що викликається за допомогою поліморфної змінної.

**Поняття поліморфної змінної**

Однією з найбільш цікавих особливостей об'єктно-орієнтованих мов програмування є той факт, що фактичний тип змінної може не збігатися з типом, заявленим при її оголошенні.

Поліморфна змінна не тільки зберігає дані фактичного типу, але і дозволяє використовувати методи фактичного типу, але й дозволяє використовувати методи фактичного типу.

Для позначення типу присвоюваного змінній при її оголошенні будемо використовувати термін "статичний тип", а для характеризування типу фактичного значення – "динамічний тип". Змінна, для якої динамічний тип може не співпадати зі статичним, називається поліморфною.

**Реалізація поліморфних змінних.**

Поліморфмою в С++ може бути лише змінна типу покажчик(\*) чи посилання(&).

Поліморфізм виникає, коли покажчику (або посилання) базового класу присвоюється покажчик на похідний клас. Правильність використання властивостей та методів динамічного типу визначає компілятор на основі статичного типу.

**Метод підстановки та покажчики**

Згідно принципу підстановки покажчику на базовий клас можна присвоїти значення адреси об’єкту будь-якого похідного класу.

**class Base**

**{public: void print(){cout << "Base" << endl;};};**

**class Derive: public Base**

**{public: void print(){cout << "Derived" << endl;} };**

**Base \*b1 = &b;**

**b1->print();**

**b1 = &d;**

**b1->print();**

**Base b;**

**Derive d;**

**b.print();**

**d.print();**

Специфікатор virtual: оголошує віртуальний метод, це означає, що:

* буде викликати метод фактичного типу (того на що у даний вказує покажчик);
* спадкується: віртуальними будуть всі методи у класах нащадках з тим же ім’ям.

Віртуальними не можуть бути статичні методи і конструктори.

***Поліморфізм при спадкуванні проявляється через комбінацію віртуальний метод + динамічна змінна екземпляр***. Приклад.

***class Base***

***{public: virtual void print(){cout << "Base" << endl;};};***

***class Derive: public Base***

***{public: void print(){cout << "Derived" << endl;} };***



**Base \*b1 = &b;**

**b1->print();**

**b1 = &d;**

**b1->print();**

**Base b;**

**Derive d;**

**b.print();**

**d.print();**

Статичне (раннє) зв’язування – це визначення чогось на етапі компіляції, наприклад змінної та її значення, функції та її виклику. Те, що визначається на етапі виконання (покажчик на функцію та адреса конкретної функції) є прикладом динамічного (пізнього) зв’язування.

Віртуальність запускає механізм пізнього зв’язування коли покажчик на об’єкт зв’язується з методом на етапі виконання за своїм фактичним типом.

**Відкладений метод**

Відкладений метод - це окремий випадок перевизначення, коли метод базового класу не має реалізації, а будь-яка корисна діяльність задається в методі дочірнього класу. У мові С ++ відкладений метод повинен бути описаний в явному вигляді з ключовим словом virtual. Тіло відкладеного методу не визначається, натомість функції «приписується» значення = 0

*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.*

*Контрольні запитання для самоперевірки*.

1. З якою метою в мові С++ реалізований поліморфізм?
2. Поясніть значення терміну “перезавантаження” функцій.
3. За допомогою яких механізмів реалізується поліморфізм?
4. Чи можна покажчику на базовий клас присвоїти значення адреси об'єкта похідного класу?
5. Що необхідно для того, щоб реалізувати поліморфізм в програмі?
6. Чим динамічний поліморфізм відрізняється від статичного?

*Контрольні запитання для надання письмових відповідей*.

1. Яким чином компілятор визначає яку з перевантажених функцій потрібно виконувати?
2. Для чого використовується перевантаження конструкторів класу? В чому переваги перевантаження конструкторів класу?