Лекція 9. Динамічна ідентифікація типів і оператори приведення типу

С ++ реалізує поліморфізм за допомогою використання ієрархії класів, віртуальних функцій і вказівників на базові класи.

Вказівник на базовий клас можна використовувати для посилання на об’єкти як базового класу, так і на об’єкти будь-якого похідного класу.

Отже, не завжди заздалегідь відомо, на об'єкт якого типу буде посилатися вказівник на базовий клас в довільний момент часу.

Це з'ясується тільки при виконанні програми – при використанні одного із засобів динамічної ідентифікації типів (run-time type identification — RTTI).

Для отримання типу об'єкта під час виконання програми використовується оператор **typeid**. Для цього необхідно включити в програму заголовок

```
#include <typeinfo>
```

Найпоширеніший формат використання оператора typeid такий:

typeid(object)

Тут елемент object означає об'єкт, тип якого потрібно отримати. Можна виконувати не тільки вбудований тип, але і тип класу, створеного програмістом. Оператор typeid повертає посилання на об'єкт типу type_info, який описує тип об'єкта object.

У класі **type_info** визначені наступні public-члени:

bool operator == (const type_info &ob);

bool operator !=(const type_info &ob);

bool before(const type_info &ob);

const char *name();

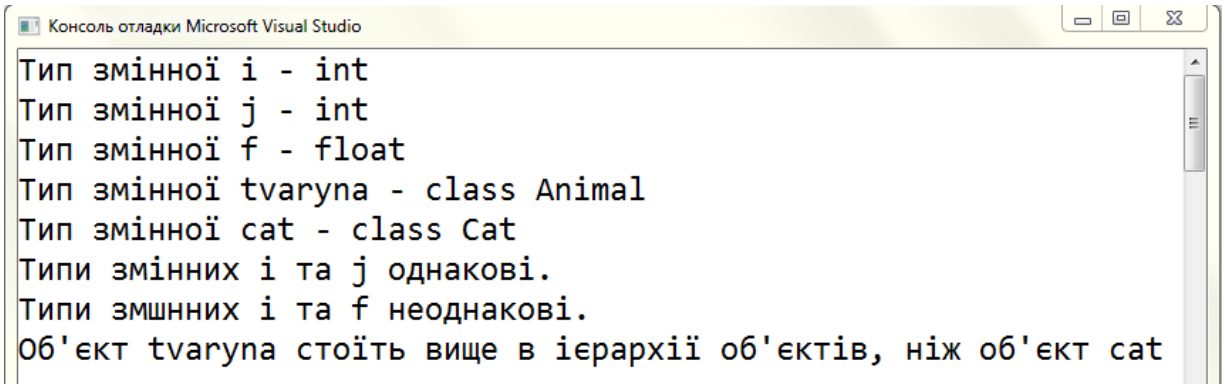
Перевантажені оператори «==» і «! =» служать для порівняння типів.

Функція before() повертає true, якщо викликаючий об'єкт стоїть вище в ієрархії об'єктів, ніж об'єкт, який використовується в якості параметра.

Розглянемо простий приклад використання оператора typeid:

```
class Animal {
 // . . .
};
class Cat : public Animal
{

};
int main()
{
 setlocale(0, "Ukr");
 int i, j;
 float f;
 Animal tvaryna;
 Cat cat;
 cout << "Тип змінної i - " << typeid (i).name() << endl;
 cout << "Тип змінної j - " << typeid (j).name() << endl;
 cout << "Тип змінної f - " << typeid (f).name() << endl;
 cout << "Тип змінної tvaryna - " << typeid (tvaryna).name() <<
endl;
 cout << "Тип змінної cat - " << typeid (cat).name() << endl;
 if (typeid (i) == typeid (j))
      cout << "Типи змінних i та j однакові." << endl;
 if (typeid (i) != typeid (f))
      cout << "Типи змшнних i та f неоднакові." << endl;
 if (typeid (tvaryna).before(typeid (cat)))
 {
      cout << "Об'єкт tvaryna стоїть вище в";
      cout <<" ієрархії об'єктів, ніж об'єкт cat" << endl;
 }
}
```

При виконанні програма генерує ось такі результати:



Якщо оператор typeid застосовується до вказівника на поліморфний базовий клас, він автоматично повертає тип реального об'єкта, на який той вказує: будь то об'єкт базового класу або об'єкт класу, виведеного з базового.

Отже, оператор typeid можна використовувати для динамічного визначення типу об'єкта, що адресується вказівником на базовий клас.

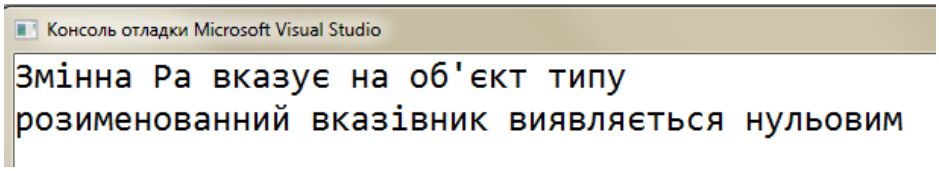
Слід пам'ятати, що вказівник обов'язково повинен бути розіменований. Коли цей розіменованний вказівник виявляється нульовим, з’являється спеціальний виняток bad_typeid.

Приклад застосування оператора typeid до ієрархії поліморфних класів:

```
class Animal {
 // . . .
public:
 virtual void fun() {} // робимо клас Animal  поліморфним
};
class Cat : public Animal
{ };
class Bear : public Animal
{ };

int main() {
 setlocale(LC_ALL, "ukr");
 Animal *Pa, tvaryna;
 Cat cat;
 Bear bear;
 Pa = &tvaryna; // Pa = NULL;
 //ПАМ'ЯТКА: Вказівник обов'язково повинен бути розименован.
 //коли цей розименованний вказівник виявляється нульовим,
 //створено спеціальний виняток bad_typeid
 try {
       cout << "Змінна Pa вказує на об'єкт типу ";
       cout << typeid (*Pa).name() << endl;
       Pa = &cat;
       cout << "Змінна Pa  вказує на об'єкт типу ";
       cout << typeid (*Pa).name() << endl;
       Pa = &bear;
       cout << "Змінна Pa  вказує на об'єкт типу ";
       cout << typeid (*Pa).name() << endl;
 }
 catch (...)
 {
 cout << endl << "розименованний вказівник виявляється нульовим ";
 }
}
```

При виконанні програма генерує ось такі результати:

У випадку, коли `Pa = NULL;` результат роботи програми буде таким:



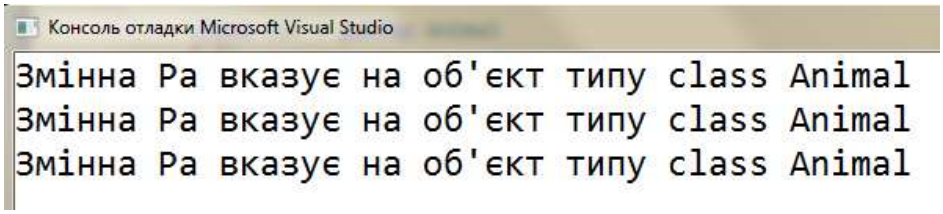
Якщо оператор typeid застосовується до вказівника на базовий клас поліморфного типу, тип реально адресуємого об'єкта, як підтверджують ці результати, буде визначено під час виконання програми.

У всіх випадках застосування оператора typeid до вказівника на не поліморфну ієрархію класів буде отримано вказівник на базовий тип, тобто те, на що цей вказівник реально вказує, визначити не можна.

У попередній програмі в базовому класі Animal приберемо ключове слово virtual:

```
class Animal {
    // . . .
public:
    virtual void fun() {} // робимо клас Animal не поліморфним
};
```

Після цього програма генерує ось такі результати:



Посилання на об'єкти ієрархії поліморфних класів працюють подібно вказівникам. Якщо оператор typeid застосовується до посилання на поліморфний клас, він повертає тип об'єкта, на який воно реально посилається, і це може бути об'єкт не базового, а похідного типу. Описаний

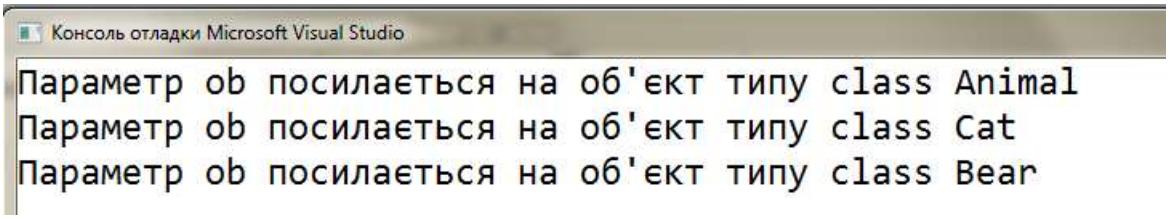
засіб найчастіше використовується при передачі об'єктів функціям за посиланням.

Наприклад, в наступній програмі функція WhatType () оголошує параметр-посилання на об'єкти базового класу Animal. Це означає, що функції WhatType () можна передавати посилання на об'єкти типу Cat або Bear або посилання на об'єкти будь-яких класів, похідних від базового класу Animal. Оператор typeid, застосований до такого параметру, поверне реальний тип об'єкта, переданого функції.

Наступний приклад демонструє застосування оператора typeid до посилального параметру:

```
class Animal {
 // . . .
public:
 virtual void fun(){} // робимо клас Animal поліморфним
};
class Cat : public Animal
{ };
class Bear : public Animal
{ };

// застосування оператора typeid до посилального параметру:
void WhatType(Animal & ob)
{
 cout << "Параметр ob посилається на об'єкт типу ";
 cout << typeid (ob).name() << endl;
}

int main()
{
 setlocale(0, "Ukr");
 Animal tvaryna;
 Cat cat;
 Bear bear;
 WhatType(tvaryna);
 WhatType(cat);
 WhatType(bear);
}
```

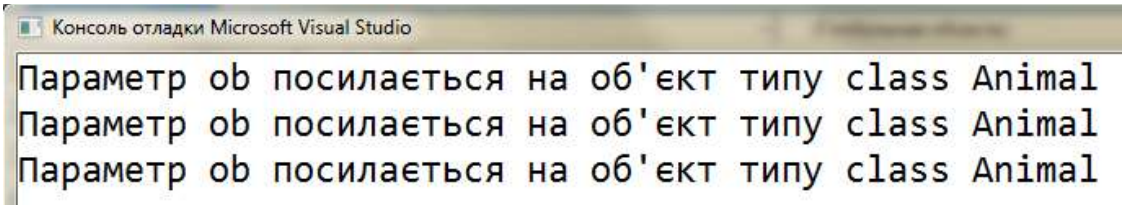
При виконанні програма генерує ось такі результати:



Після внесення в код зміни:

```
class Animal {
 // . . .
public:
 void fun() {} // робимо клас Animal не поліморфним
};
```

Програма генерує ось такі результати:



Оператори приведення типів. Оператор dynamic_cast

Оператор **dynamic_cast** виконує операцію приведення поліморфних типів під час виконання програми. Загальний формат застосування оператора dynamic_cast такий:

dynamic_cast <type> (expr)

Тут елемент type означає новий тип, який є метою виконання цієї операції, а елемент expr- вираз, що приводиться до цього нового типу. Тип type повинен бути представлений вказівником або посиланням, а вираз expr має приводитися до вказівника або посилання.

Таким чином, оператор dynamic_cast можна використовувати для перетворення вказівника одного типу у вказівник іншого або посилання одного типу на посилання іншого.

Цей оператор в основному використовується для динамічного виконання операцій приведення типу серед поліморфних типів.

Розглянемо простий приклад. Припустимо, що клас Animal - поліморфний, а клас Cat – похідний від класу Animal.

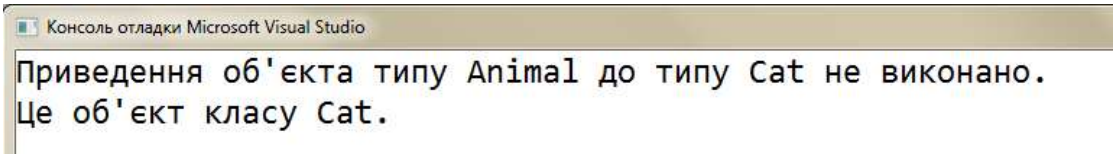
```
class Animal
{
 virtual void f() {}; // робимо клас поліморфним
};


class Cat : public Animal
{
public:
 void CatOnly() {
      cout << "Це об'єкт класу Cat."<<endl;
 }
};

int main()
{
 setlocale(0, "Ukr");
 Animal *Pa, animal;
 Cat *Pc, cat;
 // Використання оператора dynamic_cast
 Pa = &animal;
 Pc = dynamic_cast <Cat *> (Pa); // помилка
 if (Pc)
      Pc->CatOnly();
 else
   cout << "Приведення об'єкта типу Animal до типу Cat не
виконано.";
 Pa = &cat;
 Pc = dynamic_cast <Cat *> (Pa);
 if (Pc)
      Pc->CatOnly();
 else
   cout << endl << "Помилка, приведення типу має бути
реалізовано!";
}
```

В наступному фрагменті коду

```
 Pa = &animal;
 Pc = dynamic_cast <Cat *> (Pa); // помилка
```

спроба зробити операцію приведення типу буде невдалою, оскільки Pa в дійсності вказує на об'єкт базового класу Animal, і неправомірно приводити вказівник на базовий клас до типу вказівника на похідний, якщо адресується їм об'єкт не є насправді об'єктом похідного класу.

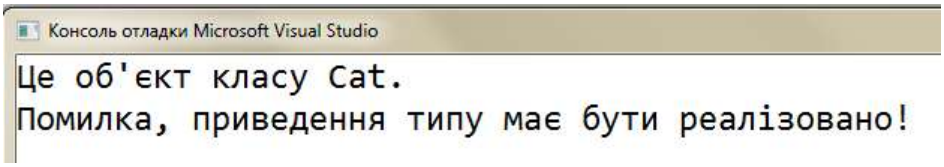
Маємо такий результат:



Якщо змінимо код:

```
Pa = &cat; // &animal; тепер вказує на об’єкт похідного класу
Pc = dynamic_cast <Cat *> (Pa);

if (Pc)
	Pc->CatOnly();
else
	cout << "Приведення до типу Cat не виконано." << endl;
 Pa = &animal; //а тут навпаки тепер вказує на об’єкт похідного класу
```

Результат буде таким:



Як бачите, використання оператора dynamic_cast спрощує логіку, необхідну для перетворення вказівника на базовий клас в вказівник на похідний клас. Ось як виглядають результати виконання цієї програми.

Резюме

1. У C++ вказівник на базовий клас також можна використовувати для посилання на об'єкт будь-якого класу, виведеного з базового.
2. Подібно вказівниками, посилання на базовий клас також можна використовувати для доступу до об'єкта похідного типу.
3. Використовуючи вказівник на похідний клас, не можна отримати доступ до об'єкта базового типу.
4. Віртуальна функція – це функція, яка оголошується в базовому класі з використанням ключового слова virtual і перевизначається в одному або декількох похідних класах. Прототипи віртуальної функції і її перевизначень повинні бути абсолютно однаковими.
5. Клас, який включає віртуальну функцію, називається поліморфним класом.

6. Якщо похідний клас не перевизначає віртуальну функцію, то використовується функція, певна в базовому класі.

7. Суто віртуальна функція - це віртуальна функція, яка не має визначення в базовому класі.

8. Клас, який містить хоча б одну чисто віртуальну функцію, називається абстрактним, у такого класу не може бути об'єктів.

9. Для отримання типу об'єкта під час виконання програми використовується оператор typeid. Якщо оператор typeid застосовується до вказівника на базовий клас поліморфного типу, тип реально адресуємого об'єкта буде визначено під час виконання програми.

10. Оператор dynamic_cast виконує операцію приведення поліморфних типів під час виконання програми. оператор dynamic_cast можна використовувати для перетворення вказівника одного типу у вказівник іншого або посилання одного типу на посилання іншого.