Лекція 6

Тема лекції: Віртуальні функції. Приклад використання віртуальних функцій. Використання конструкторів та деструкторів базових та похідних класів. «Чисті» віртуальні функції. Абстрактні базові класи. Віртуальні деструктори. Множинна спадкоємність. Використання віртуальних базових класів

Віртуальні функції

Віртуальна функція - це функція, виклик якої залежить від типу об’єкту. У класичному програмуванні необхідно було задавати тип під час написання коду. В об’єктно-орієнтованому програмуванні можна писати віртуальні функції так, щоб об'єкт визначав, яку функцію необхідно викликати, під час виконання програми. Тобто за допомогою віртуальних функцій об'єкт визначає свої власні дії. Техніку використання віртуальних функцій називають ще поліморфізмом.

Відповідно до правил C++ покажчик на базовий клас може посилатися на об'єкт цього класу або на об'єкт будь-якого іншого класу, похідного від базового.

Це правило - ключ до розуміння того, як використовуються віртуальні функції. Розглянемо низку наступних класів: *A, B, C*, де клас *B* виводиться з класу *A*, а *C* - з *B*. В програмі об'єкти класів *A*, *B*, *C* можуть оголошуватись за допомогою таких операторів:

```
A aObject;
B bObject;
C cObject;
```

За визначенням покажчик типу *A* може посилатися на будь-який з трьох об'єктів, оскільки вони пов'язані відношенням успадкування.

Наприклад:

```
A *p;
p = &cObject;
```

Тут привласнюється адреса об’єкта *cObject* покажчику на базовий клас.

Незважаючи на те, що покажчик *p* має тип *A**, а не *C**, він може посилатися на об'єкт типу *C*, тому що клас *C* є похідним від класу *A*. Ця залежність об'єктів і покажчиків працює тільки в одному напрямку.

Цей принцип стає особливо важливим, коли в зв'язаних родинними відношеннями класах визначаються віртуальні функції. Функції мають такий самий вигляд і програмуються так само, як і звичайні функції, тільки їхні оголошення випереджуються ключовим словом *virtual*. Наприклад, клас *A* може оголосити віртуальну функцію *vf()*:

```
class A
{
 public:
  virtual void vf();
};
```

Віртуальна функція може оголошуватись з параметрами, вона може повертати значення, як і будь-яка інша функція. У класі може оголошуватись стільки віртуальних функцій, скільки буде потрібно. Вони можуть бути відкритими, захищеними і закритими членами класу. У класі *B*, похідному від *A*, також можна оголосити віртуальну функцію з тим самим ім'ям. Це - функція, що заміщає. Запишемо оператор, що викликає віртуальну функцію *vf()* для об’єкту, на який посилається p:

```
p -> vf();
```

Покажчик *p* може зберігати адресу об’єкту типу *A*, *B* або *C*. Отже, під час виконання програми цей оператор викликає віртуальну функцію класу об’єкту, на який посилається *p*. Той самий оператор викликає функцію класу поточного об’єкту - дія, визначена під час виконання.

Припустимо, створюється графічна програма і розробляються різні класи для таких фігур, як квадрати, кола, лінії. Всі ці класи взаємозалежні (всі вони - фігури), тому можна спочатку створити базовий клас *Shape*:

```
class Shape
{
 public:
  virtual void Draw();
};
```

Ще невідомо, які типи фігур будуть потрібні, можливо, будуть оголошені нові фігури. Необхідно десь зберігати колекцію об'єктів *Shape*, мабуть, у масиві покажчиків. Нехай він оголошений глобальним і, отже, всі покажчики ініціалізовані нульовим значенням.

```
Shape *picture[100];
```

Кожен елемент масиву - покажчик типу *Shape**, що може посилатися на об'єкт типу *Shape* або на об'єкт будь-якого похідного від нього класу. Оскільки в класі *Shape* оголошена віртуальна функція *Draw()*, можна запрограмувати цикл, що викликає функцію *Draw()* для покажчиків із масиву *picture*.

```
int i=0;
while(i<100 && picture! =0)
{
  picture[i]->Draw();
  i++;
}
```

Отже, написаний код, який малює картинку до того, як з'явилася хоча б одна існуюча фігура. Цей код можна скомпілювати і зберегти в бібліотеці або в .OBJ-файлі. Пізніше можна вивести нові класи з класу *Shape*, зберегти об'єкти цих класів у масиві *picture* і цикл буде викликати віртуальні функції *Draw()* цих об'єктів для малювання відповідних зображень. Цей код не потребує точного завдання типів даних об'єктів, на які посилаються покажчики *picture*, потрібно тільки, щоб ці об'єкти були похідними від *Shape*. Об'єкти визначають під час виконання, яку функцію *Draw()* треба викликати. Виведемо класи *Circle* і *Line* із класу *Shape*.

```
class Circle:public Shape
{
 public:
  virtual void Draw();
};
class Line:public Shape
{
 public:
  virtual void Draw();
};
```

Тепер потрібно запрограмувати функції, що заміщують *Draw()* у класах *Circle* і *Line* для малювання кола та лінії. При виконанні програми цикл, що малює, викликає необхідні віртуальні функції, виходячи з тіла об’єкту, який адресується. У масив *picture[]* можна додавати фігури так:

```
picture[0] = new Circle;
picture[1] = new Line;
picture[2] = new Triangle;
```

Неможливо вгадати, які види об'єктів буде містити масив *picture[]*, але це не має значення для циклу малювання. Цей цикл спроможний намалювати будь-який об'єкт класу, похідного від класу *Shape*. Виклики віртуальних функцій-членів компонуються під час виконання програми за допомогою техніки, що називається пізнім зв'язуванням. Виклики звичайних функцій-членів компонуються під час компіляції (раннє зв'язування). У технічній реалізації, адреси віртуальних функцій-членів зберігаються у віртуальній таблиці. Пошуки потрібної адреси віднімають час і впливають на швидкодію програми. Віртуальні функції-члени успадковуються похідними класами, як і звичайні функції-члени. В останньому з похідних класів у лінії успадкування, можна видалити ключове слово *virtual*.

Чисті віртуальні функції

Чиста віртуальна функція - порожнє місце, що очікує свого заповнення похідним класом. Оголошення чистої віртуальної функції в класі завершується = 0. Наприклад:

```
class AbstractClass
{
 public:
  virtual void f1(void);
  virtual void f2(void) = 0;
};
```

f2() - чиста віртуальна функція-член класу. Компілятору не буде потрібно реалізації функції-члена *f2()*, на відміну від інших віртуальних функцій-членів, оголошених у класі.

Якщо клас містить хоча б одну чисту віртуальну функцію, він називається абстрактним класом. Абстрактний клас - це схема, на підставі якої створюються похідні класи. В C++ не можна створювати об'єкти, що мають типи абстрактних класів. Наприклад, компіляція такого рядка:

```
AbstractClass myObject;
```

видасть повідомлення про помилку: “Cannot create an instance of class AbstractClass” - “Не можу створити об'єкт класу AbstractClass”.

Інші функції-члени цього класу можуть викликати чисту функцію-член.

```
void AbstractClass::f1(void)
{
...
 f2();
...
}
```

Щоб використовувати абстрактний клас, треба вивести з нього новий клас:

```
class MyClass:public AbstractClass
{
 public:
  virtual void f2(void);
}
```

Похідний клас *MyClass* успадковує чисту віртуальну функцію-член, але оголошує її без знаків “=0”. У іншому місці програми треба реалізувати *f2()*:

```
void MyClass::f2(void)
{
...
}
```

Звернення до початкової чистої віртуальної функції-члена тепер переадресовані *MyClass::f2()*. За допомогою оголошень чистих віртуальних функцій-членів у похідному класі можна вставляти заглушки до коду, який може бути викликаний іншими функціями-членами. Тепер, коли усі віртуальні функції-члени реалізовані, компілятор зможе сприйняти об'єкти типу *MyClass*:

```
MyClass myObject;
```

```
myObject.f1();
```

Другий з операторів призведе до виклику успадкованої *f1()*, що викликає *f2()* із *MyClass*.

Чисті віртуальні функції можуть бути відкритими, але найчастіше вони є захищеними, оскільки передбачається, що функція буде реалізована в похідному класі.

Віртуальні деструктори

Віртуальні деструктори зазвичай застосовуються у випадках, коли в деякому класі необхідно видалити об'єкти похідного класу, на які посилаються покажчики на базовий клас. Розглянемо клас, що запам'ятовує рядкове значення:

```
class TBase
{
 private:
  char *sp1;
 public:
  TBase(const char *s)
  {
   sp1=strdup(s);
   }
 virtual ~TBase()
 {
 delete sp1;
 }
};
```

Конструктор класу *TBase* виділяє простір для рядка, звертаючись до функції *strdup()* і зберігає адресу нового рядка в покажчику *sp1*. Віртуальний деструктор звільняє цю область, коли об'єкт типу *TBase* виходить з області бачення.

Виділяємо новий клас із *TBase*:

```
class TDerived:public TBase
{
```

```
 private:
  char *sp2;
 public:
  TDerived(const char *s1,const char *s2):TBase(s1)
  {
    sp2 = strdup(s2);
  }
  virtual ~TDerived()
  {
   delete s2;
  }
};
```

Новий клас зберігає рядок, на який посилається *s2*. Новий конструктор викликає *TBase()*, передаючи рядок до базового класу, а також виділяє пам'ять для другої копії рядка, що видаляється деструктором класу.

Коли об'єкт *TDerived* виходить з області дії, важливо, щоб обидві копії рядка були видалені. Припустимо, що покажчику на клас *TBase* привласнена адреса об’єкту *TDerived*:

```
TBase *pBase;
pBase = new TDerived(“String1”,”String2”);
          ...
delete pBase;
```

При звільненні пам'яті компілятор викликає потрібний деструктор. У даному випадку покажчик *pBase* посилається на об'єкт похідного класу, тобто викликається деструктор класу *TDerived*. Об'єкти самі визначають, який деструктор викликати, тому що деструктори оголошені віртуальними. Якщо деструктори не віртуальні, викличеться деструктор базового класу, залишивши другу копію рядка в пам'яті.

Конструктори викликаються тільки при створенні об’єкта, а тип об’єкта, що створюється, повинен бути відомий компілятору. Тому конструктор не може бути віртуальним.

Конструктори не бувають віртуальними, але можуть викликати віртуальну функцію. А для руйнування об’єкту тип не важливий, тому можливі віртуальні деструктори.

Множинна спадкоємність. Віртуальні базові класи

При множинній спадкоємності похідний клас може мати декілька базових класів.

Для виведення нового класу з декількох базових треба перерахувати імена базових класів після імені нового:

```
class D:public A, public B, public C
 {
    ....
 };
```

Специфікатори доступу можна використовувати різні й у довільному порядку.

Похідний клас успадковує всі властивості всіх базових класів. Якщо в двох базових класах оголошується функція-член із тим самим ім'ям, то для доступу до них використовується оператор розширення області бачення.

Якщо класи *A*, *B*, *C* мають конструктори за замовчуванням, то у похідному класі *D* їх можна викликати так:

```
class D:public A, public B, public C
{
 public:
  D():A(), B(), C() { }
};
```

Конструктор можна реалізувати окремо, в іншому місці програми:

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

Конструктори викликаються в тому порядку, в якому оголошені їхні базові класи.

В прямому ациклічному графі (ПАГ) успадкування члени класу можуть з'являтися неодноразово. Розглянемо ПАГ множинної спадкоємності:

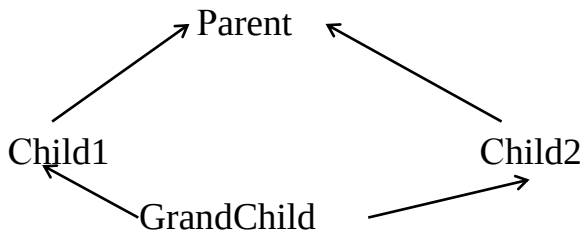

Рис. 2.1 Прямий ациклічний граф множинної спадкоємності

Елементи класу *Parent* з'являються двічі в класі *GrandChild*. Перший набір успадковується через *Child1*, другий набір - через *Child2*. Таке успадкування буває небажаним. Віртуальні базові класи забезпечують механізм для запобігання дублювання елементів у класі.

```
class Parent {  };
class Child1: public virtual Parent {... .};
class Child2: public virtual Parent {... .};
class GrandChild: public Child1, public Child2 {... .};
```

Тут клас *GrandChild* багаторазово успадковує поля класу Parent. Завдяки використанню віртуальних базових класів, об'єкти класу *GrandChild* мають по одному полю даних з базового класу. Віртуальні базові класи ініціалізуються перед будь-якими невіртуальними базовими класами й у тому порядку, у якому вони оголошені. Якщо віртуальний базовий клас має хоча б один конструктор, то він повинен мати конструктор за замовчуванням. З похідного класу не можна прямо викликати параметризований конструктор віртуального базового класу.