Object Oriented Programming with C++

2024 Spring Semester

21 CST H3Art

Chapter 9 Polymorphism and Virtual Functions

- **Polymorphism（多态性）**
  - One name, multiple forms
  - In C++，it mean **different functions** with the **same function name**.
  - In OOP, it means **objects belonging to different classes（不同类的对象）** are able to **respond to the same message（回应相同的信息）**, but in different forms.
- **Function overloading（函数重载）** is a kind of polymorphism (**early binding（早绑定）** or **static binding（静态绑定，编译阶段确定）**).
  - Overload member functions in one class:

    ```
    show(int, char);
    show(char*, float);
    ```

  - Overload member functions of a base class in a derived class:
    - By matching arguments
    - Using `::`
- C++ supports a more flexible mechanism **virtual function（虚函数）** to achieve **run time polymorphism（运行时多态）**: i.e. select the appropriate member function while the program is running. The process is termed **late binding（迟绑定）** or **dynamic binding（动态绑定，运行时确定）**.
- Three questions, for public inheritance，can we:
  - ☑ Assign a derived class object to a base class object?
  - ☑ Use a base class object reference to refer to a derived class object?
  - ☑ Use a base class object pointer to point to a derived class object?
- `class A` ⟵ `class B`, suppose we have the following code part:

  ```
  A * p ; // pointer refer to class A

  A A_obj ;
  B B_obj ;

  p = & B_obj ; // p refer to object of class B
  ```

  - Using `p`，the `public` **members** of `B_obj` which are inherited **from** `class A` **can be accessed（只有基类`A`的成员可以被访问）**，but **NOT** the members defined by `class B` (unless explicitly cast `p` to `A` type)
  - Using pointer `p` to base class, no matter `p` refer to base class object or derived class, `object.p->func()` **always executes the function defined in the base class**.
  - To execute different version of the functions, we need to use objects explicitly:

    ```
    first_obj.func();
    second_obj.func();
    ```

- **Achieving Polymorphism（实现多态性）**
  - Run time polymorphism is achieved only when a **virtual function** is accessed through **a pointer to the base class**.
    （指向基类对象的指针，其调用了基类的虚函数时会唤起多态性）

- The **prototypes** of the base class version of a virtual function and all the derived class versions must be **identical**. **（基类的虚函数和派生类的函数必须有相同的函数原型，这样才能启用多态性）**
- if a virtual function is defined in the base class, it **need not be necessarily redefined in the derived class**. In such cases, calls will **invoke the base function**. **（基类的虚函数不一定需要在派生类中被实现，此时根据多态性，调用的会是基类的函数）**
- Example:

```
class base {
public:
  virtual void vf1();
  virtual void vf2();
  virtual void vf3();
  void f();
};

class derived : public base {
public:
  void vf1();    // virtual function
  void vf2(int); // function overloading
  char vf3();    // error
  void f();      // function overloading
};
```

- **Virtual Destructor（虚析构函数）**
  - Declare the destructor of a base class as virtual function, the destructors of **all the classes derived from the base class become virtual functions（注意：派生类继承的函数也全都变成虚函数，但不用加`virtual`关键字）**, although the names of the destructors are different from that of the base class!
  - Cases when the destructor must be virtual: in a class system derived from a base class, if **dynamic create object** is needed, the **destructor must be virtual**, to achieve polymorphism while deleting objects. **（需要动态创建对象时，若该对象所属的类是派生类，它的基类析构函数必须是虚函数，以此实现多态析构）**
  - **Normally the destructor of a base class is declared as virtual（通常析构函数被定义为虚函数，无论是否需要自析构，但这可以确保其派生类完成析构）**. Even when the base class do not need self defined destructor, define an empty virtual destructor, to make sure the derived object will be destroyed properly.
- **Pure Virtual Function（纯虚函数）**
  - Definition: A virtual function declared in a base class that **has no definition relative to the base class（没有定义函数体）**. Such functions are called "do–nothing" functions.
  - Pure virtual functions provide **public interfaces（公有接口）** for derived classes.
  - Syntax of declaring pure virtual function:

```
class class_name{
    ...
    virtual type function_name(arglist) = 0;
};
```

    there is **no function body**, but **not empty function body**. **（没有函数体的定义，但是函数体并不为空？有点绕，总之看上面的例子）**
    Assigning `0` to function name, is **equivalent to assign null to the pointer refers to the function body（与赋值`NULL`到指向函数体的函数指针是等价的）**. The function can not be invoked before it is redefined in the derived classes.
  - **Abstract class（抽象类）**: class which contains **at least one pure virtual functions**. **（类内包含至少一个纯虚函数）**
    - An abstract class is not used to create objects, it can only be used as base class. **（只用于创建基类，不用于创建对象）**
    - Abstract class can be used to declare pointers and references. **（用于声明指针和引用）**

- Example:

```
class point {
  /*...*/
};

class shape { // 抽象类
  point center;

public:
  point where() { return center; }
  void move(point p) {
    center = p;
    draw();
  }
  virtual void rotate(int) = 0; // 纯虚函数
  virtual void draw() = 0;      // 纯虚函数
};

class abs_circle: public shape {
  int radius;

public:
  void rotate(int) {};
  // abs_circle::draw() is still a pure virtual function if it is undefined in class abs_circle
  // therefore, class abs_circle is still an abstract class
}

int main() {
  shape x;            // error，抽象类不能建立对象
  shape *p;           // ok，可以声明抽象类的指针
  shape f();          // error，抽象类不能作为返回类型
  void g(shape);      // error，抽象类不能作为参数类型
  shape &h(shape &); // ok，可以声明抽象类的引用
  return 0;
}
```

- **Polymorphism Instance（多态实例）**

```cpp
#include <iostream>

using namespace std;

class Number {
public:
  Number(int i) {
    val = i ;
  }
  virtual void Show() = 0;

protected:
  int val;
};

class Hextype: public Number {
public:
  Hextype(int i): Number(i) {}
  void Show () {
    cout << hex << val << endl; // hex和下面的dec是后面涉及到的I/O库定义的函数
  }
};

class Dectype: public Number{
public:
  Dectype(int i): Number(i) {}
  void Show() {
    cout << dec << val << endl;
  }
};

void fun(Number &n) {
  n.Show();
}

int main() {
  Dectype d(50);
  fun(d); // d.Show();
  Hextype h(16);
  fun(h); // h.Show();
  return 0;
}
```