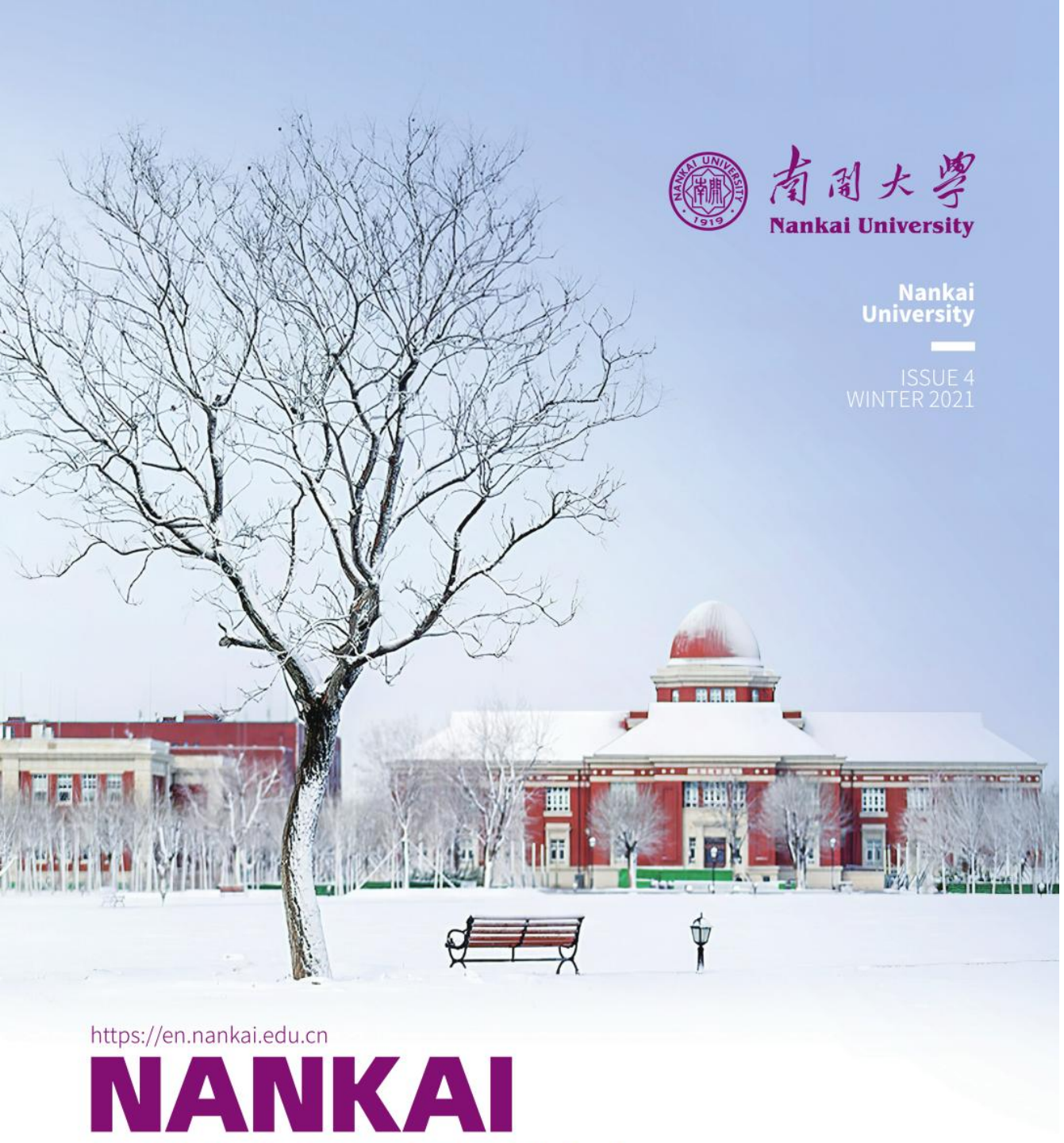

09-多态性与虚函数

龚成 南开大学软件学院

cheng-gong@nankai.edu.cn

函数超载与虚函数

函数超载

虚函数

函数超载

- C++允许在不同的类中出现其**原型完全相同的函数**，即所谓的函数超载。
- 仅在基类与其派生类的范围内实现（与使用）。
- 允许（支持）多个不同函数（的实现）使用完全相同的函数名、函数参数表以及函数返回类型。
- 通常这些函数是语义相近的或完全相同的（仅具体实现方法即实现代码不同）。

函数超载

- 基类和派生类中的函数超载可能存在二义性。
- 根据虚函数定义，函数超载二义性可由虚函数处理。

虚函数

- 在定义某一基类（或其派生类）时，若将其中的某一个非静态成员函数的属性说明为 virtual，则称该函数为虚函数。其一般说明形式为：

  virtual〈返回类型〉〈函数名〉（〈参数表〉）{…}；

- 虚函数的使用与函数超载密切相关。若基类中某函数被说明为虚函数，则意味着其派生类中也要用到与该函数同名、同参数表、同返回类型，但函数（实现）体不同的同一个超载函数。
- 在编译阶段系统无法判断此次虚函数调用应执行哪一段函数代码。只有到了运行过程执行到此处时，才能临时判断应执行哪一段函数代码，对虚函数的这种处理方式称为**动态联编（dynamic banding）**。

虚函数

- 虚函数的机制要点为：
  - 在基类 CB中说明某一函数成员 f()为虚函数，方法是在**说明前加关键字“virtual”**。如 virtual void draw();
  - 在 CB的各个派生类 CD1，CD2，…，CDn 中定义与f()的原型完全相同（但函数体可以各异）的函数成员 f()，无论是否用关键字 virtual来说明，它们**都自动地被定义为虚函数**，即**派生类中虚函数处的关键字 virtual可以省略，但基类处不可省**。
  - 当在程序中采用**以 CB 类指针 pb 的间接形式调用函数 f()，即使用 pb->f()时，系统对其将采用动态联编的方式进行处理**。

虚函数

- 虚函数的实现原理：
  - 虚函数是C++类中的成员函数，它允许派生类提供特定的实现。当一个类中至少有一个虚函数时，编译器会为这个类创建一个虚函数表。
  - 虚函数表是一个存储函数指针的数组，每个包含虚函数的类都有一个。虚函数表中存储了类中所有虚函数的地址。
  - 当通过基类指针或引用调用一个虚函数时，程序实际上使用的是动态联编。编译器生成的代码首先会查找对象内部的虚函数表指针，然后根据这个指针找到虚函数表，并在表中查找对应的函数指针来执行。（这意味着又虚函数的类对象会大一些）
  - 由于虚函数表的存在，程序可以在运行时识别对象的实际类型，即使对象是通过基类类型的指针或引用访问的，即使通过基类指针调用函数，如果对象实际上是派生类的一个实例，也会调用派生类中覆盖的虚函数版本，因为基类指针调用的虚函数表指针其实是指向派生类的虚函数表的，而不是基类的虚函数表。
  - 虚函数表在程序启动时由编译器生成，并存储在程序的只读数据段中。每个包含虚函数的对象都会有一个虚函数表指针，指向这个类的虚函数表。

虚函数

```
class graphelem
{ // 自定义类 graphelem
protected:
    int color; // 颜色 color
public:
    graphelem(){color = 0;}
    virtual void draw()
    {
     cout <<"graphelem"<< endl;
    }; // 基类中含有一个虚函数
};
```

```
class line : public graphelem
{ // 自定义类 line，为基类 graphelem 的派生类
    public: // 虚函数 draw 负责画出“line”
    virtual void draw() { cout << "draw line" << endl; };
};
class circle : public graphelem
{ // 自定义类 circle，为基类 graphelem 的派生类
    public: // 虚函数 draw，负责画出“circle”
    virtual void draw(){ cout << "draw circle" << endl; };
};
class triangle : public graphelem
{ // 类 triangle，为基类 graphelem 的派生类
    public: // 虚函数 draw 负责画出“triangle”
    virtual void draw(){ cout << "draw triangle" << endl; };
};
```

虚函数

```
int main()
{
    graphelem *p[3]; // 定义一个指针数组，数组元素为基类 graphelem 的指针
    p[0] = new line(); // 指针数组的第一个元素指向派生类 line 的对象
    p[1] = new circle(); // 指针数组的第二个元素指向派生类 circle 的对象
    p[2] = new triangle(); // 指针数组的第三个元素指向派生类 triangle 的对象
    for (int i = 0; i < 3; i++)
        p[i]->draw(); // 调用指针数组中各个元素的 draw() 函数
    return 0;
}
```

虚函数

- 虚函数和重载函数的区别和联系：
  - 它们都是在程序中设置一组**同名函数，都反映了面向对象程序中的多态性特征**；
  - 虚函数**不仅同名，而且同原型**；
  - 虚函数**仅用于基类和派生类之中**，不同于函数重载，可以是类内或类外函数；
  - 虚函数在程序运行时**动态联编以确定具体函数代码**，而**重载函数在编译时即可确定**。
  - 虚函数**一般是一组语义相近的函数，而函数的重载，相互之间可能是语义无关的**。
  - **构造函数不能说明为虚函数，但是构造函数通常有许多重载函数**。这是因为构造函数的调用一般出现在对象创建的同时或之前，这时无法用指向其对象（尚未创建）的指针来引用它。
  - **析构函数可以说明为虚函数，此时这一组虚函数的函数名是不同的**。当在析构函数中采用基类指针释放对象时，应注意把析构函数说明为虚函数，以确定释放的对象。

纯虚函数与抽象基类

纯虚函数与抽象基类

- 如果不准备在基类的虚函数中做任何事情，则可在虚函数的原型后加上“=0”字样替掉函数定义体（没有具体的实现），则这样的虚函数称为纯虚函数。

virtual <函数原型>**= 0;**

- 纯虚函数只为其派生类的各虚函数规定了一个一致的“原型规格”，该虚函数的实现将在它的派生类中给出。

纯虚函数与抽象基类

- **含有纯虚函数的基类称为抽象基类。注意，不可使用抽象基类来说明并创建它自己的对象**，只有在创建其派生类对象时，才有抽象基类自身的实例伴随而生。
- 抽象基类中的虚函数使基类作为这一组类的抽象，**基类成为它的若干派生类的对外接口**。通过抽象基类，再“加上”各派生类的特有成员以及对基类中那一纯虚函数的具体实现，可以构成一个具体的实用类型。
- 许多引入虚函数的程序，把基类的虚函数说明为纯虚函数，从而使基类成为一种抽象基类，可以更自然地反映实际应用问题中对象之间的关系。
- **如果一个抽象基类的派生类中没有定义基类中的纯虚函数**，而只是继承了基类的纯虚函数，则这个**派生类还是一个抽象基类**，其中仍包含着继承而来的那个纯虚函数。