抽象类

概念

- **纯虚函数**是虚函数后面加`= 0`
- 至少含有一个纯虚函数的类称为**抽象类**
- 抽象类**不能**实例化对象
- 纯虚函数声明中不能含有函数体，但可以放在类外
- 纯虚析构函数**必须**有函数体
- 如果子类没有重写父类的纯虚函数，则子类仍然是抽象类

```
class TestA {
    ...
    virtual void foo() = 0; // 标准要求不能在此实现foo，依赖于编译器实现
    ...
}
```

应用

1. 规范化派生类的接口
2. 用于定义某些不适合生成对象的父类

override与final

override

`override`关键字：说明性关键字，标识此函数为子类重写父类的虚函数，只能用于虚函数

```
class TestB : public TestA {
    ...
    virtual void foo() override     // 标识此函数为子类重写父类的虚函数
    {
        ...
    }

    virtual void bar()
    {
        ....
    }
    ...
}
```

final

`final`关键字：标明此函数不能再被子类给重写，只能用于虚函数

```
class TestA {
    ...
    virtual void foo() final    // 标明此函数不能再被子类给重写
    {
        ...
    }
    ...
}
```

多重继承

语法

- 子类会继承所有父类的数据（数据成员与成员函数）

```
class A {
    ...
}

class B {
    ...
}

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

内存分布

- 先父类，后子类
- 父类的内存排列顺序受继承顺序影响

指针转换

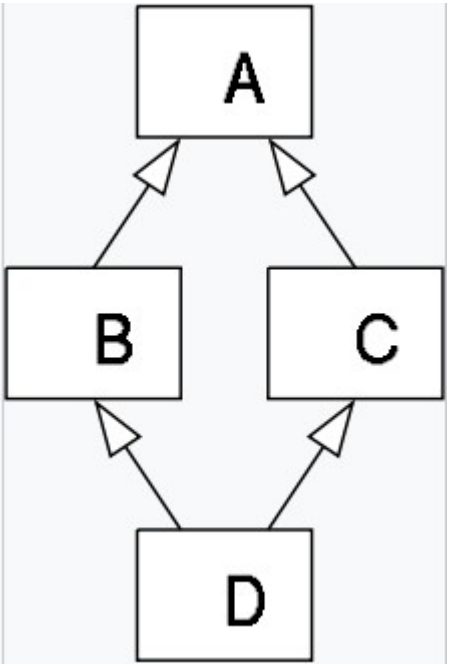
- 当子类对象赋值给父类指针（或引用）的时候，会自动转换为父类位于子类内存中的首地址
- `this`指针也会发生类似的转换

构造析构的顺序

- 先构造父类，后构造子类
- 顺序与继承顺序有关
- 析构与构造的顺序相反

菱形继承

菱形继承的问题：子类包含多份父类的数据




```
class A {
    ...
}

class B : public A {
    ...
}

class C : public A {
    ...
}

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

虚继承

解决菱形继承的问题

语法

- 虚继承中虚基类的数据只有一份，存放在对象的末尾
- 所有的中间父类类都要使用`virtual`关键字

```
class A {
    ...
}

class B : virtual public A {
```

```
    ...
}

class C : virtual public A {
    ...
}

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

构造析构的顺序

- 先构造虚基类，再构造中间父类，最后子类
- 析构的顺序和构造相反

```
{
    D d;    // 构造 A::A() -> B::B() -> C::C() -> D::D()
}           // 析构 D::~D() -> C::~C() -> B::~B() -> A::~A()
```