1.1 OOP: 概述

面向对象程序设计的核心思想是数据抽象、继承和动态绑定。

1. 通过使用数据抽象，我们可以将类的接口和实现分离。
2. 使用继承，可以定义相似的类型并对其相似关系建模。
3. 使用动态绑定，可以在一定程度上忽略相似类型的区别，而以统一的方式使用它们的对象。

对于某些函数，基类希望它的派生类各自定义适合自身的版本，此时基类就将这些函数声明为虚函数。

在C++语言中，当我们使用基类的引用或指针调用一个虚函数时将发生动态绑定。

基类通常都应该定义一个虚析构函数，即使该函数不执行任何实际操作也是如此。

在派生类对象中含有与基类对应的组成部分，这一事实是继承的关键所在。

派生类构造函数

尽管在派生类对象中含有从基类继承而来的成员，但是派生类并不能直接初始化这些成员。和其他创建了基类

每个类控制它自己的成员初始化过程。

除非我们特别指出，否则派生类对象的基类部分会像数据成员一样执行默认初始化。如果想使用他的基类构造

首先初始化基类的部分，然后按照声明的顺序依次初始化派生类的成员。

派生类使用基类的成员

派生类可以访问基类的公有成员和受保护成员。

继承与静态成员

如果基类定义了一个静态成员，则在整个继承体系中只存在该成员的唯一定义。不论从基类中派生出来多少个

```
class Base{
    public:
      static void statmem();
};

class Deirved : public Base{
    void f(const Derived &);
};
```

静态成员遵循通用的访问控制规则，如果基类中的成员是private的，则派生类无权访问它。假设某个静态成员

```
void Derived::f(const Derived &derived_obj){
    Base::statmem();
    derived_obj.statmem();
    statmem();
}
```

派生类的声明

派生类的声明于其他类差别不大，声明中包含类名但是不包含它的派生列表。

```
class Bulk_quote : public Quote; //
class Bulk_quote;  //
```

被用作基类的类

如果我们想将某个类用作基类，则该类必须已经定义而非仅仅声明。

```
class Quote;
class Bulk_quote : public Quote {
    ...
};  //
```

防止继承的发生

有时我们会定义这样一种类，我们不希望其他类继承它，或者不想考虑它是否合适作为一个基类。为了实现这

```
class NoDerived final {....}; // NoDerived
class Base{...};
class Last final : public Base {....}; // Last
```

1.2.3 类型转换与继承

理解基类和派生类之间的类型转换是理解C++语言面向对象编程的关键所在。

我们可以将基类的指针或引用绑定到派生类对象中。可以将基类的指针或引用绑定到派生类对象上有一层极为

和内置指针一样，智能指针也支持派生类向基类的类型转换，这意味着我们可以将一个派生类对象的指针

静态类型和动态类型

当我们使用存在继承关系的类型时，必须将一个变量或其他表达式的静态类型与该表达式表示的动态类型区分

基类的指针或引用的静态类型可能与其动态类型不一致，我们一定要理解其中的缘由。

不存在从基类向派生类的隐式类型转化

在对象之间不存在类型转换

派生类向基类的自动类型转换只对指针或引用类型有效，在派生类型和基类类型之间不存在这样的转换。

当我们用一个派生类对象为一个基类对象初始化或赋值时，只有该派生类对象中的基类部分会被拷贝、移

1.3 虚函数

通常情况下，如果我们不使用某个函数，则无须为该函数提供定义。但是我们必须为每一个虚函数都提供定义

对虚函数的调用可能在运行时才被解析

当某个虚函数通过指针或引用调用时，编译器产生的代码直到运行时才能确定应该调用哪个版本的函数。被调

当我们通过一个具有普通类型(非引用非指针)的表达式调用虚函数时，在编译时就会将调用的版本确定下来。

当我们在派生类中覆盖了某个虚函数时，可以再一次使用virtual关键字指出该函数的性质。然而这么做并非

一个派生类的函数如果覆盖了某个继承而来的虚函数，则它的形参类型必须与被它覆盖的基类函数完全一致。

基类中的虚函数在派生类中隐含地也是一个虚函数。当派生类覆盖了某个虚函数时，该函数在基类中的形

final和override说明符

派生类如果定义了一个函数与基类中虚函数的名字相同但是形参不同，这仍然是合法的行为。编译器将认为新

要想调试并发现这样的错误显然非常困难。在C++11标准中，我们可以使用override关键字来说明派生类中的

我们还能把某个函数指定为final，如果我们已经把函数定义成final了，则之后任何尝试覆盖该函数的操作者

虚函数与默认实参

和其他函数一样，虚函数也可以拥有默认实参。如果某次函数调用使用了默认实参，则该实参由本次调用的静

如果虚函数使用默认实参，则基类和派生类中定义的默认实参最好一致。

回避虚函数的机制

在某些情况下，我们希望对虚函数的调用不用进行动态绑定，而是强迫其执行虚函数的某个版本。使用作用域

```
double undiscounted = base->Quote::net_price(42);
```

通常情况下，只有成员函数(或友员)中的代码才需要使用作用域运算符来回避虚函数的机制。

什么时候我们需要回避虚函数的默认机制呢？通常是当一个派生类的虚函数调用它覆盖的基类的虚函数版本时

如果一个派生类虚函数需要调用它的基类版本，但是没有使用作用域运算符，则在运行时该调用将被解析

1.4 抽象基类

纯虚函数

我们通过在函数体的位置书写=0就可以将一个虚函数说明为纯虚函数。其中，=0只能出现在类内部的虚函数声

```
class Disc_quote : public Quote{
    public:
      Disc_quote() = default;
      Disc_quote(const std::string &book, double price, std::size_t qty, double disc):
      double net_price(std::size_t ) const = 0; //
    protected:
      std::size_t quantity = 0;
      double discount = 0.0;
};
```

值的注意的是，我们也可以为纯虚函数提供定义，不过函数体必须定义在类的外部。也就是说，我们不能在类

含有纯虚函数的类是抽象基类

含有纯虚函数的类是抽象基类。抽象基类负责定义接口，而后续的其他类可以覆盖该接口。我们不能创建一个

我们不能创建抽象基类的对象。

派生类构造函数只初始化它的直接基类

每个类控制其对象的初始化过程。因此，即使派生类没有自己的数据成员，它也仍然需要像原来一样提供一个

1.5 访问控制与继承

每个类分别控制自己的成员初始化过程，与之类似，每个类还分别控制着其成员对象对于派生类来说是否可访
受保护的成员

`protected`说明符可以看作是`public`和`private`