序

面向对象语言编译器提供了很多服务，如构造函数，析构函数，虚函数，继承，多态..等

这本书解释编译器对我们写的C++代码做了什么手脚

目前所有编译器对虚函数的实现都是使用各个class专属的virtual table，大小固定，在程序执行前就构造好了。

第一章：关于对象

类的对象占用的存储空间：只有非静态的成员数据加上数据在内存中对齐而填补的空间加上指向虚函数表的虚指针占用存储空间。

C++未选用的两种对象模型：简单对象模型，表格驱动对象模型。

C++对象模型：每一个对象内部都有非静态数据成员。静态数据成员，静态成员函数和非静态成员函数不在每一个对象内部，只在外部存在一份。虚函数分两步，第一步每一个类的每一个虚函数都有一个指向它的指针，所有虚函数的指针放在一张表格中，虚表（vtbl），第二步每一个类的对象安插一个指针，指向相关的虚表，虚指针（vptr），这时每个类的对象的模型中存着它的非静态数据成员和一个vptr，vptr指向vtbl，vtbl中的第一个槽存储的是类的type\_info

C++对象模型加上继承：1、bptr，也就是base class table模型，但是添加了间接性，继承了几层，就要间接几层。2、在vtbl中加入base类的指针。

C++三种编程范式：程序模型，抽象数据模型，面向对象模型。

C++支持多态：基类指针结合虚函数实现多态；dynamic\_cast实现基类指针转为派生类指针。

指针类型：所有指针都是一样，大小一样，而指针类型决定了指向的地址怎么处理，如int的指针，int占4个字节，那么指针指向的地址开始连续四个字节属于这个int对象。而void指针只有地址没有空间，因为void类型没有空间。

Cstruct的C++合理用途：传递一个复杂的class object的全部或者部分到某个c函数去时，struct可以将数据封装起来，并保证拥有和c兼容的空间布局。

加上多态：一个指针或者引用之所以支持多态，是因为它们不引发内存中任何“与类型有关的内存委托操作”，会受到改变的，只有他们所指向内存的大小和内容解释方式而已。

第二章：构造语义函数学

编译器会合成默认构造函数的四种情况：1、类包含一个有默认拷贝构造函数的类的对象成员，编译器会合成默认构造函数，其内容会是被包含对象的构造函数的调用，对于自己类其他成员并不初始化，自己写的默认构造函数初始化其他成员之后编译器或扩展这个函数并在其中添加被包含类对象调用默认构造函数。如果包含多个类的对象作为成员，那这个类的默认构造函数会按照声明的顺序依次调用它们类的默认构造函数，如果这类没写默认构造函数，就默认调用，如果写了，就按照写的调用，然后安插被包含对象的类的默认构造函数调用代码进用户自己的构造函数中。2、如果一个没有任何构造函数的类派生于一个带有默认构造函数的基类，那派生类的默认构造函数就被编译器认为是有必要的，会被构造出来，它将调用上一层基类的默认构造函数3、带有一个虚函数的类，这是为了正确的初始化每一个类对象的vptr，如果没有声明构造函数，就会生成一个。4、派生自虚类或者虚类的子类。

调用拷贝构造函数的三种情况：1、赋值 2、按值进行传参 3、调用函数返回结果。当一个类对象以另一个同类实例作为初值，拷贝构造函数就会被调用。对于1只有初值才调用拷贝构造函数，不是初值调用拷贝赋值运算符。没有显式声明拷贝构造函数时，会进行逐位拷贝。

编译器会合成默认拷贝构造函数的情况：1、类成员中有类对象，而此对象的类包含拷贝构造函数2、类继承的基类存在拷贝构造函数，无论是显式声明的还是编译器合成的3、类声明了虚函，合成的拷贝构造函数会显示设定vptr的应有的值4、派生自继承串联中，有一个或多个虚继承基类

如果按照位拷贝可以不出现问题，就不用显式的声明拷贝构造函数，这样效率更高

NRV优化：简单的讲，传递引用，而非传递值，对引用直接操作，替代返回值

第三章：Data语义学

虚继承：只会出现一份实例的继承，在多重继承中应用，不会出现重复的拷贝

空类对象大小为一字节

空类虚继承空类为4字节，子类对象包含一个指向基类对象的指针，而不包含基类类似的一字节

类对象大小为三部分：类本身成员，编译器自动加上的数据成员（一般为支持虚函数加入的vptr），边界调整

现在的编译器一般把vptr放在类对象的前端

多重派生，也是按照拼接的方式将不同基类的成员和vptr都保留下来，成为一个新的模型。这样各个派生类的vptr指向自己的虚表，从而实现多态

类似& 类名：：类成数据成员 的操作是获取这个数据成员在类对象模型中的偏移量-

第四章：function语义学

类的非静态成员函数：编译器会转化成一种非成员函数，对象调用相应非静态成员函数的结果会变成对象作为参数传递给转化后的非成员函数

类的虚函数被调用：会在内部转化为类的对象得到vptr并找到在虚表中相应位置的函数然后传递调用的对象作为参数给这个虚函数

静态成员函数：无this指针，因此不能直接存取其类中的非静态成员，不能被声明为const，volatile，virtual。不需要对象就能被调用

inline函数：在循环中编译器会将被视为不变的表达式提到循环之外，大幅提升效率。

指向成员函数的指针：提取一个成员数据的地址，得到的是在类对象模型中的偏移量，加上类对象的地址就能得到完整的地址。取一个成员函数的地址，得到的是内存中真正的地址，但是也不够完全，需要对象才能调用，需要对象的this指针。

inline函数扩展期间：每一个形式参数都会被对应的实际参数取代，不会有传参过程，而如果由中间表达式或函数会生成临时对象存储中间结果避免重复计算。因此效率高。

inline函数局部对象：以单一表达式扩展多次，则每次扩展都需要自己的一组局部变量，如果以分离的多个式子被扩展多次，那么只需一组局部变量，就可以重复使用。

第五章：构造、析构、拷贝语义学

对于抽象类：含有纯虚函数的类，不可能有实例。当该类拥有一个数据成员时，还是需要有一个显式的构造函数，否则它的派生类的这个数据成员将无法决定初值，一般来说，数据成员的初始化应该在构造函数或者在成员函数中。否则会失去封装性

纯虚析构函数：继承关系中的每一个类都必须定义它，因为每一个派生类析构函数都会被编译器加以扩张，以静态方式调用每一个虚基类和上一层基类的析构函数，如果缺乏任何一个基类析构函数，就会链接失败。

不要把派生类中不会改变的函数定义为虚函数，这样效率反而会变低。

虚函数中const的使用：如果派生类一定要修改一个数据成员时，const就会带来困扰，因此虚函数最好别用const

抽象类声明：有至少一个纯虚函数，析构函数不要是纯虚的，纯虚函数不要用const限定，如果有数据成员应该有protected属性的构造函数。

变量生命周期：局部变量在局部代码执行完后销毁，全局变量与程序生命相同，new创建的堆变量则直到delete运算符摧毁为止。全局对象在定义时调用构造函数，在程序执行exit时调用析构函数，而new和delete会在内部转化为执行码

许多函数以传值的方式返回一个局部对象变量，那么为这个对象的类添加一个拷贝构造函数比较合理。

构造函数和析构函数在继承中，派生类产生对象时从最顶层的基类调用构造函数，销毁对象时从派生类调用析构函数

第六章 执行期语义学

C++不太容易从程序源码看出表达式的复杂度，程序执行期会发生一些转换

对象的构造和析构：析构必须在每一个离开点执行

new：用c的malloc来实现的，先分配对象，然后给值

delete：总是用c的free来实现的