高级程序设计

https://github.com/qc2105/ExercisesCppHowToProgram9thEd

github上有本书的所有代码，链接如上。

第九章 类的深入刨析：抛出异常

9.2

包含保护

在#ifndef和#endif中的内容不会被重复定义。

用大写头文件名，并用_代替.

黏性设置

setfill是黏性设置，一旦指定，后面都会起作用。setw非黏性，只对最接近的输出起作用。默认填充字符出现在数字之前。

成员函数

在类外定义的成员函数依然属于类内的作用域。在类内定义的成员函数被隐式定义为内联（inline）的。但内联只是建议，编译器保留不内联的权利。

初始化

在C++11前，只有const static int 数据成员可以在类中声明的地方进行初始化。在C++11中，可以使用类内的初始化符在类定义中声明的位置初始化任何数据成员。

抛出异常

一个类函数可以抛出异常（如invalid_argument）来指出无效数据

9.3 类的作用域和类成员的访问

在类的作用域内，类的成员可以被类的所有成员函数直接访问，也可以通过名字引用。在类外，public类成员可以通过对象的句柄（handle）使用。句柄可以是对象名称，对象引用，对象指针。

类作用域和块作用域

成员函数中的变量具有块作用域，当函数定义了和类作用同名的变量时，小范围的块作用域生效。被隐藏的类作用域变量可以通过在其名前加类名和二元作用域分辨运算符：：访问

成员选择运算符

原点运算符（.)前面加对象名称或者引用，箭头运算符（->)前面加对象的指针。

9.4 访问函数和工具函数

访问函数用来读取或者显示数据，也可以用来测试条件真假。常常称这样的函数为判定函数。如判断是否为空，是否满。

工具函数是用来支持类的其他成员函数操作的private成员函数。

9.5 Time类实例研究：具有默认实参的构造函数。

见程序

使用列表初始化器调用构造函数

C++11可以使用列表初始化器来初始化任何变量。

Time t2{2} 和 Time t2={2}都是正确的语法。

重载的构造函数和委托构造函数

C++11允许构造函数调用一个类中的其他构造函数。这样的构造函数被称为委托构造函数（delegating constructor）。

9.6 析构函数

不接受任何值，也不返回任何值.当对象被撤销时被隐式调用。实际上析构函数不释放对象占用的内存空间。

9.7 何时调用构造函数和析构函数

全局作用域

全局作用域的对象的构造函数在所有函数（包括main）开始执行之前调用。当main函数执行结束时，相应的析构函数被调用。exit和abort函数可以使函数立刻结束，不执行自动对象的析构函数。

局部对象

当运行到对象定义处开始构造，当程序离开作用域时，相应的析构函数被调用。

static局部对象

static局部对象的构造函数会在程序第一次执行到定义处时执行一次，析构函数在main函数结束时。

所有的对象存储在栈内。析构的顺序与构造的顺序相反。

9.8 Time 实例研究 返回private数据成员的引用或指针

private可以以引用的形式被类的非private函数返回，因此会破坏类的封装性，需要避免。

9.9 默认的逐个成员赋值

赋值运算符（=）可以将一个对象赋给另一个类型相同的对象。

9.10 const对象和const成员函数

通过const关键字声明某些常量。修改const的命令会在编译时报错，而不是程序运行时。

对于const对象，编译器只允许const类型成员函数调用。一个成员函数要在两处同时声明const。一处是函数原型的参数列表后插入，一处是函数定义时在函数体开始的左括号之前。const成员函数调用同一类的同意实例的非const成员函数也会编译错误。

析构函数和构造函数因都需修改对象所以不能被声明为const。但构造函数依然可以初始化const对象。在构造函数中调用非const成员函数来作为初始化const对象的一部分时允许的。

9.11 组成：对象作为类的成员

见代码。

一个类可以将其他类的对象作为其成员，称为组成

组成是传值还是引用不是看传入成员初始化器的是否是引用，而是看在定义组成时是否是引用。比如书上的代码，如果在Employee.h中将birthDate改为Date&类型，那么析构函数只会执行一次。而书上虽然传入的是引用，但是数据成员是按传值方式复制的。所以析构函数会执行2次。

9.12 friend函数和friend类

友元关系是单向授予的，既不对称也不传递。

友元不是成员函数

9.13 this指针

每个对象都有一个this指针来访问自己的地址。this指针不是对象本身的一部分，this指针作为隐式的参数被传递给对象的每个非static成员函数。

this指针的类型取决于对象的类型和使用this指针的成员函数的类型。在非const成员函数中this指针为class * const指向对象的值可以被修改，但是所指向的对象不变。const成员函数中，类型为const class * const。

在成员函数中 x，this->x,(*this).,这三种等效。括号是必须的，因为圆点运算符(.)优先级高于键址符（*）

可以令成员函数返回*this来连续赋值

9.14 static类成员

static数据成员不会被拷贝，仅有一份副本供类的所有对象共享。

C++11中类内初始化能允许在类定义中变量申明的位置初始化他。如果static成员对象有默认构造函数。

即使没有任何类的对象存在时。类的static数据成员和成员函数仍然存在。当没有对象存在时想要访问public static成员，可以使用域作用符。

类的static需要被所有访问文件的客户代码使用，所以不能在.cpp文件中将他们声明为static，而在.h中将它们声明为static。

static成员函数没有this指针，将static成员函数声明为const是一个编译错误。

第十章 操作符重载 （考试重点）

本章展示了如何使c++的操作符能够与用户定义的对象一起工作——这个过程称为操作符重载。

操作符作为成员函数重载时，它们必须是非静态的。因为静态函数没有this指针，不能知道是哪个类的对象进行的调用。

不能被重载的操作符

```
.      .*       ::       ?:
```

重载时不能改变操作符的优先级，结合性，元数。

当重载()、[]、->或任何赋值操作符时，必须将操作符重载函数声明为类成员。

二元操作符可以重载为带有一个形参的非静态成员函数，也可以重载为带有两个形参的非成员函数(其中一个形参必须是类对象或对类对象的引用)。

以<举例，当重载为非静态成员函数时，x<y会被处理为x.operator<(y)。当被重载为非成员函数时，x<y被处理为operator<(x,y)。

当重载流运算符时，会将返回值设为引用来支持连续调用比如cout<<a<<b

流运算符重载一般为非成员函数，因为类的对象一般是操作的右值。否则会出现class_object >> cin 这样的语句。

类的一元操作符可以重载为不带参数的非静态成员函数，也可以重载为带一个参数的非成员函数，该参数必须是类的对象(或对对象的引用)。

一元操作符，如!可以作为带有一个形参的非成员函数重载。

自增和自减操作符的前缀和后缀版本都可以重载。

要重载自增操作符以允许同时使用前缀和后缀自增，每个重载的操作符函数必须具有不同的签名，以便编译器能够确定要使用哪个版本的++。

前缀版本完全像任何其他前缀一元操作符一样被重载。

假设我们想要在Date对象d1中的日期上加1。

当编译器看到表达式++d1时，编译器会产生如下调用 d1.operator++()。

此操作符函数的原型为: Date &operator++()

如果前缀自增运算符实现为非成员函数，当编译器看到表达式++d1时，编译器生成函数调用 operator++(d1)。

该操作符函数的原型在Date类中声明为: Date &operator++(Date &);

重载后缀自增操作符是一个挑战，因为编译器必须能够区分重载的前缀和后缀自增操作符函数的签名。

c++中采用的约定是，当编译器看到表达式d1++时，它调用 d1.operator++(0)

此函数的原型为: Date operator++(int)

参数0是严格意义上的“哑值”，使编译器能够区分前缀和后缀增量操作符函数。

相同的语法用于区分前缀和后缀自减运算符函数

```
    friend Date& operator++(Date& d)
    {
        d.day++;
        return d;
    }
    friend Date operator++(Date& d, int)
    {
        Date temp = d;
        d.day++;
        return temp;
    }
```

```
Date& operator++()
{
    this->day++;
    return *this;
}

Date operator++(int)
{
    Date temp = *this;
    this->day++;
    return temp;
}
```

前缀运算符返回对象的引用，后缀运算符返回对象的值。因此前缀运算符支持连续调用，后缀运算符不支持。同时后缀运算符优先级高于前缀运算符。所以++day++会造成编译错误。

你可以控制程序中对象和任何内置类型或用户定义类型的数组的内存分配和释放。称为动态内存管理;使用new和delete执行。

new操作符为Date类型的对象分配适当大小的存储空间，调用默认构造函数初始化该对象，并返回指向new操作符右侧指定类型的指针(即Date *)。

Date *ptr=new Date();如果new无法在内存中为对象找到足够的空间，则通过“抛出异常”表示发生了错误。

要销毁动态分配的对象，使用delete操作符如下:delete ptr;

该语句首先调用ptr所指向对象的析构函数，然后释放与该对象关联的内存，将内存返回给自由存储区

注意：析构函数并不提供释放内存的功能

等号出现在对象的声明时，不是赋值，而是调用拷贝构造函数。将等号后面的值作为参数传入构造函数。

当operator[]被重载时，传入下标可以不是整数。

当Array类的对象超出作用域时调用析构函数

无论操作符函数是作为成员函数实现还是作为非成员函数实现，操作符在表达式中的使用方式都是相同的。

当操作符函数作为成员函数实现时，最左边(或唯一)的操作数必须是操作符类的对象(或对对象的引用)。

如果左操作数必须是不同类或基本类型的对象，则该操作符函数必须实现为非成员函数(就像我们在10.5节重载<<和>>分别作为流插入操作符和提取操作符时所做的那样)。

如果非成员操作符函数必须直接访问该类的私有成员或受保护成员，则可以将其设为类的友元。

特定类的操作符成员函数只有当二元操作符的左操作数是该类的特定对象，或者一元操作符的单操作数是该类的对象时才会被调用。

转换构造函数(conversion constructors)——可以用单个参数调用的构造函数(我们将其称为单参数构造函数),可以将其他类型的对象(包括基本类型)转换为特定类的对象。

转换操作符(conversion operator)(也称为强制类型转换操作符(cast operator))可用于将一个类的对象转换为另一个类型。这样的转换操作符必须是非静态成员函数。

除了拷贝构造函数之外，任何可以用单个参数调用且未显式声明的构造函数都可以被编译器用于执行隐式转换。

防止使用单参数构造函数进行隐式转换,我们在每个单参数构造函数之前声明关键字explicit的原因是为了在通过转换构造函数进行不正确的隐式转换时抑制这种转换。

声明为显式的构造函数不能用于隐式转换。

从c++ 11开始，与显式声明单参数构造函数类似，可以显式声明转换操作符，以防止编译器使用它们执行隐式转换。

如explicit MyClass::operator char *() const;

第十一章 面对对象编程：继承

继承是is-a关系，组成是has-a关系。

虚函数不能是静态函数