## 何时选用成员函数还是友元函数来重载运算符？

判断标准：

1. C++ 规定，箭头运算符->、下标运算符[ ]、函数调用运算符( )、赋值运算符=只能以成员函数的形式重载。
2. 不需要进行类型转换或参数具有逻辑对称性使用成员函数
3. 除以上情况使用友元函数。

## 何时使用explicit关键字来修饰构造函数？

尽量使用

## this指针是什么含义？

this 实际上是成员函数的一个形参，一个const指针，指向当前对象。

## 什么时候该显式地调用析构函数？

1. to destroy an object that was constructed using [placement-new](https://en.cppreference.com/w/cpp/language/new).
2. to destroy an object through an allocator member function such as [std::allocator::destroy()](https://en.cppreference.com/w/cpp/memory/allocator/destroy).
3. to destroy an object that was constructed through the allocator.

## void function (type\_Name const&)是什么意思？

**void function (type\_Name const&)** is equivalent to **void function (const type\_Name &)**

当对引用添加 const 限定后，编译器允许引用绑定到类型不一致的数据。当引用的类型和数据的类型不一致时，如果它们的类型是相近的，并且遵守「数据类型的自动转换」规则，那么编译器就会创建一个临时变量，并将数据赋值给这个临时变量（这时候会发生自动类型转换），然后再将引用绑定到这个临时的变量。当引用的类型和数据的类型不遵守「数据类型的自动转换」规则，那么编译器将报错，绑定失败。将引用类型的形参添加 const 限制的潜在好处有三个：

* 使用 const 可以避免无意中修改数据的编程错误；
* 使用 const 能让函数接收 const 和非 const 类型的实参，否则将只能接收非 const 类型的实参；
* 使用 const 引用能够让函数正确生成并使用临时变量。即引用类型可以与被引用对象不一致。

## 函数声明后=delete是什么意思？

If, instead of a function body, the special syntax = **delete** ; is used, the function is defined as deleted. Any use of a deleted function is ill-formed (the program will not compile). 可以使得拷贝构造函数和赋值运算符禁用。如下例：

thread\_guard(thread\_guard const&)=delete;

thread\_guard& operator=(thread\_guard const&)=delete;

## 返回值中Type()是什么意思？

Nothing is returned from the constructor. The syntax Type() is not a constructor call, it creates a temporary object of type A (and calls the constructor in the process).

## What is cv-qualifier?

Const and volatile.

Volatile 是一种类型修饰符，提醒编译器他后面所定义的变量随时都有可能改变，因此编译后的程序每次需要存储或读取这个变量的时候，都会直接从变量地址中（内存中）读取数据。如果没有volatile关键字，则编译器可能优化读取和存储，可能暂时使用寄存器（缓存）中的值，如果这个变量由别的程序更新，将出现不一致。

volatile 关键字是一种类型修饰符，用它声明的类型变量表示可以被某些编译器未知的因素更改，比如：操作系统、硬件或者其它线程等。遇到这个关键字声明的变量，编译器对访问该变量的代码就不再进行优化，从而可以提供对特殊地址的稳定访问。

一般说来，volatile用在如下的几个地方：

* 1) 中断服务程序中修改的供其它程序检测的变量需要加 volatile；
* 2) 多任务环境下各任务间共享的标志应该加 volatile；
* 3) 存储器映射的硬件寄存器通常也要加 volatile 说明，因为每次对它的读写都可能由不同意义；

## What is decay type?

The decay type of T is the same type that results from the standard conversions that happen when an lvalue expression is used as an rvalue, with its cv-qualifier stripped.

## What is the process of initialization constructor?

Construct a [thread](https://www.cplusplus.com/thread) object that represents a new [joinable](https://www.cplusplus.com/thread::joinable) thread of execution.  
The new thread of execution calls *fn* passing *args* as arguments (using [decay copies](https://www.cplusplus.com/decay) of its *lvalue* or *rvalue* references).  
The completion of this construction synchronizes with the beginning of the invocation of this copy of *fn*.

## static 关键字的含义?

静态变量具有全局变量的生命周期，但只能作用于自己的作用域。

静态成员函数为类服务而不是为某一个类的具体对象服务。

静态全局变量在声明它的整个文件都是可见的，而在文件之外是不可见的。

静态函数不能被其它文件所用。

静态局部变量始终驻留在全局数据区，直到程序运行结束。

## decltype 的含义？

decltype 是 [C++](http://c.biancheng.net/cplus/)11 新增的一个关键字，它和 auto 的功能一样，都用来在编译时期进行自动类型推导。

## typedef 的用法？

4)为复杂的声明定义一个简单的别名  
如：typedef int (\*A) (char, char);A是我们定义的别名，表示的是一个指向函数的指针，该函数有两个char类型的参数，返回一个int类型的值。则A类型的对象可以指向任何符合上述规则的函数。

## 什么是函数指针？

函数指针就是一个指向函数的指针。函数本身不能作为形参，同样的，函数也不能作为返回值，但是我们可以通过指针的方式来调用函数。例如：

void for\_each(T\* begin, T\* end, const Func& f)；

一般的函数指针可以这么定义:  
　　**int (\*func)(int,int);**

事实上,为了代码的移植考虑,一般使用typedef定义函数指针类型.

typedef int (\*FUN)(int,int);

## 成员函数后加const是什么意思？

这种函数只能读取成员变量的值，而不能修改成员变量的值。

## 无名结构体的特点

指定了一个无名的结构体类型，它没有名字。显然不能再以此结构体类型去定义其他变量。

## 函数参数传递注意事项

当参数为非const引用时，不可传递右值，否则会报如下错误信息：

initial value of reference to non-const must be an lvalue.

## shared\_ptr智能指针

具体来讲，智能指针和普通指针的用法是相似的，不同之处在于，智能指针可以在适当时机自动释放分配的内存。也就是说，使用智能指针可以很好地避免“忘记释放内存而导致内存泄漏”问题出现。多个 shared\_ptr 智能指针可以共同使用同一块堆内存。并且，由于该类型智能指针在实现上采用的是引用计数机制，即便有一个 shared\_ptr 指针放弃了堆内存的“使用权”（引用计数减 1），也不会影响其他指向同一堆内存的 shared\_ptr 指针（只有引用计数为 0 时，堆内存才会被自动释放）

## unique\_ptr智能指针

作为智能指针的一种，unique\_ptr 指针自然也具备“在适当时机自动释放堆内存空间”的能力。和 shared\_ptr 指针最大的不同之处在于，unique\_ptr 指针指向的堆内存无法同其它 unique\_ptr 共享，也就是说，每个 unique\_ptr 指针都独自拥有对其所指堆内存空间的所有权。

这也就意味着，每个 unique\_ptr 指针指向的堆内存空间的引用计数，都只能为 1，一旦该 unique\_ptr 指针放弃对所指堆内存空间的所有权，则该空间会被立即释放回收。

## C++中的side effect 是什么意思？

A function or expression is said to have a side effect if it does anything that persists beyond the life of the function or expression itself.

Common examples of side effects include changing the value of objects, doing input or output, or updating a graphical user interface (e.g. enabling or disabling a button).

## Side effect就是“副作用”，通常是对于一个函数而言的，说一个函数“有副作用”或者“没有副作用”。 如果一个函数修改了自己范围之外的资源，那就叫做有副作用，反之，就是没有副作用。 用一个更具体的方式来说，假设某个函数在某种参数组合下返回一个结果N，我们现在就写死这个函数直接return N，如果这个修改完之后，整个程序一切如初，那就说明这个函数没有副作用，反之，就是有副作用。 很明显，有副作用的函数会做（不限于）这些事情： 修改全局变量 修改输入参数所引用的对象 做输入输出操作 调用其他有副作用的函数。

## C++ lambda

定义一个 lambda 匿名函数很简单，可以套用如下的语法格式：

[外部变量访问方式说明符] (参数) mutable noexcept/throw() -> 返回值类型  
{  
   函数体;  
};

## 具体有哪些情况是以拷贝的方式来初始化对象呢？

1) 将其它对象作为实参

Student stu2(stu1);

2) 在创建对象的同时赋值

Student stu2 = stu1;

3) 函数的形参为类类型

void func(Student s){

//TODO:

}

func(stu);

4) 函数返回值为类类型

Student func(){

Student s("小明", 16, 90.5);

**return** s;

}

Student stu = func();

## 面向对象， 面向过程， 函数式编程

**面向过程**

**优点：**

流程化使得编程任务明确，在开发之前基本考虑了实现方式和最终结果，具体步骤清楚，便于节点分析。

效率高，面向过程强调代码的短小精悍，善于结合数据结构来开发高效率的程序。

**缺点：**

需要深入的思考，耗费精力，代码重用性低，扩展能力差，后期维护难度比较大。

## 面向对象

**优点:**

结构清晰，程序是模块化和结构化，更加符合人类的思维方式；

易扩展，代码重用率高，可继承，可覆盖，可以设计出低耦合的系统；

易维护，系统低耦合的特点有利于减少程序的后期维护工作量。

**缺点：**

开销大，当要修改对象内部时，对象的属性不允许外部直接存取，所以要增加许多没有其他意义、只负责读或写的行为。这会为编程工作增加负担，增加运行开销，并且使程序显得臃肿。

性能低，由于面向更高的逻辑抽象层，使得面向对象在实现的时候，不得不做出性能上面的牺牲，计算时间和空间存储大小都开销很大。

**函数式编程**

相较于命令式的编程模式，函数式强调纯函数和不可变性，这就带来了计算确定性，函数式的确定性可以大大的增强代码的健壮性。但世间没有两全法，计算确定性就意味着没有副作用，而没有副作用，函数式无法完成交互类操作，如IO等。

让你的函数就像变量一样来使用

## Using的用法

使用using可以改变基类成员在派生类中的访问权限。

## Final

将类标记为final，方法时直接在类名称后面使用关键字final，如此，意味着继承该类会导致编译错误。

C++中还允许将方法标记为final，这意味着无法再子类中重写该方法。

## 函数继承

对于成员函数要引起注意，编译器仅仅是根据函数的名字来查找的，不会理会函数的参数。换句话说，一旦内层作用域有同名的函数，不管有几个，编译器都不会再到外层作用域中查找，编译器仅把内层作用域中的这些同名函数作为一组候选函数；这组候选函数就是一组重载函数。

只有一个作用域内的同名函数才具有重载关系，不同作用域内的同名函数是会造成遮蔽，使得外层函数（上层类）无效。派生类和基类拥有不同的作用域，所以它们的同名函数不具有重载关系。

事实上，通过派生类创建对象时必须要调用基类的构造函数

派生类构造函数中只能调用直接基类的构造函数，不能调用间接基类的

将派生类对象赋值给基类对象时，会舍弃派生类新增的成员，也就是“大材小用

只能用派生类对象给基类对象赋值，而不能用基类对象给派生类对象赋值

## 仿函数

仿函数（Functor）又称为函数对象（Function Object）是一个能行使函数功能的类。仿函数的语法几乎和我们普通的函数调用一样，不过作为仿函数的类，都必须重载 operator() 运算符。因为调用仿函数，实际上就是通过类对象调用重载后的 operator() 运算符。

## 迭代器(iterator)

是一中检查容器内元素并遍历元素的数据类型，

|  |  |
| --- | --- |
| begin() | 返回指向容器中第一个元素的迭代器。 |
| end() | 返回指向容器最后一个元素所在位置后一个位置的迭代器，通常和 begin() 结合使用。 |

Dereference 是什么意思？

**Dereferencing** a pointer means taking the address stored in a pointer and finding the value the address points to.  Dereferencing does not get rid of a reference, it is indirection. Dereferencing means to act upon the object a pointer points to.

## C++模板特化与偏特化

#### Void\*

当void\*指针作为函数的输入和输出时，表示可以接受任意类型的输入指针和输出任意类型的指针

对主版本模板类、全特化类、偏特化类的调用优先级从高到低进行排序是：全特化类>偏特化类>主版本模板类。这样的优先级顺序对性能也是最好的。

#### =default

每当我们声明一个有参构造函数时，编译器就不会创建默认构造函数。

默认函数需要用于特殊的成员函数（默认构造函数，复制构造函数，析构函数等）

自定义默认构造函数存在两个问题：1. 程序员工作量变大 2. 没有编译器自动生成的默认特殊构造函数效率高。

所以使用=default构造默认函数

#### string\_view

string\_view模板专用化提供了一种有效的方法，可将只读、异常安全、非拥有的句柄传递给任何类似于字符串的对象，并将其序列的第一个元素置于位置零。

#### [[noreturn]]

指定函数不会返回，编译器可以做特定优化

#### 多态与虚函数

(非虚函数)通过基类指针只能访问派生类的成员变量，但是不能访问派生类的成员函数

有了虚函数，基类指针指向基类对象时就使用基类的成员（包括成员函数和成员变量），指向派生类对象时就使用派生类的成员。

可以通过基类指针对所有派生类（包括直接派生和间接派生）的成员变量和成员函数进行“全方位”的访问，尤其是成员函数，在大型工程尤其有用，表现在基类有众多的子类。

下面是构成多态的条件：

* 必须存在继承关系；
* 继承关系中必须有同名的虚函数，并且它们是覆盖关系（函数原型相同）。
* 存在基类的指针，通过该指针调用虚函数。

什么时候使用虚函数？

首先看成员函数所在的类是否会作为基类。然后看成员函数在类的继承后有无可能被更改功能，如果希望更改其功能的，一般应该将它声明为虚函数。如果成员函数在类被继承后功能不需修改，或派生类用不到该函数，则不要把它声明为虚函数。

包含纯虚函数的类称为抽象类（Abstract Class）。之所以说它抽象，是因为它无法实例化，也就是无法创建对象。原因很明显，纯虚函数没有函数体，不是完整的函数，无法调用，也无法为其分配内存空间。  
  
抽象类通常是作为基类，让派生类去实现纯虚函数。派生类必须实现纯虚函数才能被实例化。