第1条：区分指针和引用。

指针可以为空，但引用必须有对象。C++要求初始化引用

指针一个对空进行测试。指针可以被重新赋值，而引用不行

没有引用的东西，或者改变引用的时候，用指针。总有对象可以引用的时候，用引用。

因此，当你知道你有东西要引用，而你永远不想引用任何其他东西时，引用就是选择的特征。当实现那些语法要求不需要使用指针的运算符时，它们也是合适的。在所有其他情况下，坚持使用指针。

第2条：更喜欢C++风格的强制转换。

static\_cast, 用于编译时已知的类型转换，例如基本数据类型之间的转换、指针类型之间的转换（如派生类指针转换为基类指针）。

const\_cast, 用于添加或移除const或volatile修饰符

dynamic\_cast,主要用于多态类型之间的转换（如基类指针转换为派生类指针,可反。）

reinterpret\_cast用于低级别的类型转换，例如将指针转换为整数，或将一种类型的指针转换为另一种类型的指针。

第3条：永远不要多态地对待数组。

array[i] 实际上是指针算术的简写形式，它等价于 \*(array+i)。我们知道 array 是指向数组起始位置的指针，但 array+i 指向的内存位置与 array 指向的内存位置之间有多远呢？它们之间的距离是 i \* sizeof(数组中的对象)，因为在 array[0] 和 array[i] 之间有 i 个对象。

至少这是编译器看待问题的方式。但如果你传递了一个 BalancedBST 对象数组给 printBSTArray，编译器很可能是错误的。在这种情况下，编译器会假设数组中的每个对象的大小是 BST 的大小，但实际上每个对象的大小是 BalancedBST 的大小。

请注意，如果避免在继承另一个具体类（如BST）的过程中使用具体类（如BalancedBST），则不太可能犯多态处理数组的错误。

第33条：使非叶类抽象。

Lizard liz1;

Lizard liz2;

Animal \*pAnimal1 = &liz1;

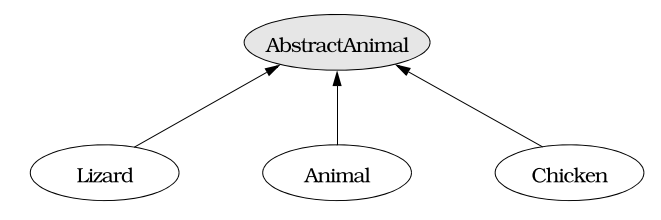
Animal \*pAnimal2 = &liz2;

...

\*pAnimal1 = \*pAnimal2;

因为\*pAnimal1是静态绑定，因此调用时候是调用animal的指针，所以只会复制一部分。

C++的多态性依赖于**函数签名的一致性**。



文本, 表格

中度可信度描述已自动生成

在一个上下文中需要抽象可能是巧合，但在多个上下文中需要抽象通常是有意义的。因此，有用的抽象是那些在多个上下文中需要的抽象。也就是说，它们对应于本身有用的类（即，具有该类型的对象是有用的），并且对于一个或多个派生类的目的也是有用的。这就是为什么从具体基类到抽象基类的转换是有用的：只有当现有的具体类将要用作基类时，它才强制引入新的抽象类，即，当类将要在新的上下文中被（重新）使用时。

**希望通过公有继承将两个具体类关联起来的愿望，通常表明需要一个新的抽象类**。

一般规则仍然适用：非叶子类应该是抽象类。

第4条：避免无意义的默认构造函数。

在这种情况下，没有默认构造函数的类将与许多模板不兼容。随着C++程序员对模板设计的了解越来越多，这个问题的重要性应该会降低。

**. 应该设置默认构造函数的情况**

表格

描述已自动生成

第5条：小心用户定义的转换函数。

图形用户界面

描述已自动生成

授予编译器执行隐式类型转换的许可证通常弊大于利，因此除非您确定需要，否则不要提供转换函数。

第6项：区分递增和递减运算符的前缀和后缀形式。

后缀运算符是先获取后递增，因此会有一个临时对象保存旧值，然后有构造和析构。前缀没有

* **前缀递增/递减：返回引用，直接修改对象。**
* **后缀递增/递减：返回值，基于前缀运算符实现，需要创建临时对象。**

第7条：切勿超载&&，||，或，。

短路运算符，前面执行后，后面不用判断。

表格

中度可信度描述已自动生成

第8项：理解new和delete的不同含义。

new操作符总是执行这两项操作，分配足够的内存，然后呼叫构造函数。

* 你可以通过添加额外的参数来重载 operator new，但第一个参数必须始终是 size\_t 类型。
* **operator new** 负责分配内存，但不涉及对象的构造。
* **new 运算符** 负责调用构造函数，将原始内存转换为对象。

Delete也是一样。

New【】和new是不同的操作符

鉴于内存管理器面临的这些需求，编译器自带的 operator new 和 operator delete 采取了一种折中策略也就不足为奇了。它们对所有人都能较好地工作，但对任何人都不是最优的。如果你对程序的动态内存使用模式有很好的理解，你通常会发现，自定义的 operator new 和 operator delete 会比默认版本表现更好。这里的“表现更好”意味着它们运行得更快——有时快几个数量级——并且需要的内存更少——最多可减少 50%。对于一些（尽管不是所有）应用程序来说，用自定义版本替换默认的 new 和 delete 是获得显著性能提升的简单方法。