关联参考　要想进一步了解如何删除调试代码，请参见第 8.6 节。

在 C++ 语言中，这个子程序将删除单个指针，但还需要实现一个类似的 SAFE_DELETE_ARRAY 子程序来删除数组。

通过将内存处理集中到这两个子程序中，还可以使 SAFE_NEW 和 SAFE_DELETE 在调试模式和生产模式中的行为有所不同。例如，当 SAFE_DELETE 在开发期间检测到试图释放一个空指针时，它可能会终止程序的运行，但在生产环境中，它可能只是简单地记录下这个错误，然后继续执行。

可以轻松地将此方案改写为 C 语言中的 calloc 和 free 以及其他使用指针的语言。

使用去指针技术　指针很难理解，还容易用错，而且相关代码往往依赖于机器并不可移植。如果能想到一个替代指针的方案，并且能合理地工作，那么请使用该方案来替代指针，以便为自己省去一些麻烦。

C++ 语言的指针

深入阅读　关于在 C++ 中使用指针的更多技巧，请参见《Effective C++ 中文版》（第 2 版和第 1 版，原英文版分别出版于 1998 年和 1996 年）。

C++ 语言引入了一些与使用指针和引用相关的特殊细节。下面介绍一些适合在 C++ 语言中使用指针的指导原则。

理解指针和引用之间的区别　在 C++ 语言中，指针 (*) 和引用 (&) 都间接指向一个对象。对于外行来说，唯一的差别似乎只是字面上的不同：object->field 和 object.field。其实，它们最重要的区别是，引用必须总是引用一个对象，而指针则可以指向空值，并且引用所引用的内容在引用初始化后不能更改。

“传址”(pass by reference) 参数使用指针，“传值”(pass by value) 参数使用 const 引用　C++ 语言向子程序传递参数的默认方式是传值而不是传址。当以传值的方式向一个子程序传递一个对象时，C++ 会创建该对象的一份拷贝，当对象被传递回调用子程序时，会再次创建一份拷贝。对于大型对象而言，这种拷贝非常费时且耗资源。因此，当向子程序传递对象时，通常会希望避免拷贝对象，这就意味着你希望传址而不是传值。

然而，有时你可能希望将“传值”的语义（即传递的对象不应该被改变）与“传址”的实现（即传递对象本身而不是拷贝）相结合。

在 C++ 语言中，这个问题的解决方法是使用指针 (pointer) 来实现“传址”，同时——虽然术语听起来很奇怪——使用“const 引用”(const reference) 来实现“传值”。下面举一个例子：