new expression、operator new 和 placement new——三个妞（new）的故事（2）

两个 delete 后的问题

最近在网上看到两个关于指针 delete 后的问题。第一种情况：

```
int* p = new int;delete p;delete p;// p 为什么能 delete 两次，而程序运行的时候还不报错。
```

第二种情况：

```
int* p = new int;delete p;*p = 5;    //delete 后对*p 进行再赋值居然也可以（他的平台上运行并没有引发什么错误）？
```

在回答这两个问题之前，我们先想想 delete p; 这一语句意味着什么？p 指向一个地址，以该 地址为起始地址保存有一个 `int` 变量（虽然该变量并没有进行初始化），`delete p` 之后 `p` 所指向的地址空间被释放，也就是说这个 `int` 变量的生命结束，但是 `p` 仍旧是一个合法的指 针，它仍旧指向原来的地址，而且该地址仍旧代表着一个合法的程序空间。与 `delete` 之前唯 一的不同是，你已经丧失了那快程序空间的所有权。这带来一个什么样的问题？你租了一间储 物室（`int* p = new int;`），后来退租了（`delete p;`），但你却保存了出入该储物室的 钥匙（指针 `p`）没有归还。拥有这片钥匙，你或许什么都不做，这自然没有问题。但是：

你或许出于好心，又跑过去告诉房东，“Hi！这储物室已经退租了（第一种情况）”。哦噢，会 发生什么？我们假设此时这个房子已经有了新的租客。愚笨的房东直接相信了你的话，认为这 个储物室空着，把它又租给新的人。于是一间只能给一个人用的储物室，却租给了两个人，再 之后各种难以预料的情况就会发生。

又或许，你很无耻，你虽然退租，但却想用你的钥匙依旧享有储物室的使用权（第二种情况），结果呢，你存在这间储物室的东西可能会被现在的租客丢掉，而你也可能把他的东西丢掉，腾 出空间来放你的。

回到上面的程序上来，毫无疑问的是上面的程序在语法上来讲是合乎规范的，但是暗藏着很大 的逻辑错误，不论你对一块已经释放的内存再度 `delete`，还是再度给它赋值，都暗含着很大 的危险，因为当你 delete 后，就代表着将这块内存归还。而这块被归还的内存很可能已经被再 度分配出去，此时不论是你再度 `delete` 还是重新赋值，都将破坏其它代码的数据，同时你存 储在其中的数据也很容易被覆盖。至于报不报错，崩不崩溃，这取决于

有一个怎么样的"房东"， 聪明且负责的"房东"会阻止你上述的行为——终止你的程序，懒惰的房东，则听之任之。

上述情况下的指针 p 被称为野指针——指向了一块"垃圾内存",或者说指向了一块不应该读写的 内存。避免野指针的好方法是，当一个指针变为野指针的时候，马上赋值为 `NULL`，其缘由在 于，你可以很容易的判断一个指针是否为 `NULL`,却难以抉择其是否为野指针。而且，`delete` 一个空指针，不会做任何操作，因此总是安全的。

不用一个基类指针指向派生类数组？

《深度探索 C++对象模型》中指出，不要用一个基类指针指向派生类的数组。因为在他的 cfront 中的 `vec_delete` 是根据被删除指针的类型来调用析构函数——也就是说虚函数机制在这 儿不起作用了。照这样的思路来说，对一个派生类的数组依次调用其基类的析构函数，显然大 多时候不能正确析构——派生类一般大于其基类。但是我感兴趣的一点是，这么多年过去了，这 样一个不太合理的设计是否有所改进呢？说它不太合理是，以 C++编程者的思路，在这样一种情 况下，它应该支持多态，而且在这种情况下支持多态并不需要太复杂的机制和代价。我在 vc++ 2008 和 vc++ 2010 下的结果是：是的，有与 cfront 不同，它支持多态。

我的测试代码如下：

```
class point{public:
    virtual ~point(){
        std::cout<<"point::~point()"<<std::endl;
    }private:
    int a;};class point3d:public point{public:
    virtual ~point3d()
    {
        std::cout<<"point3d::~point3d()"<<std::endl;
    }private:
    int b;};int main(){
    point *p=new point3d[2];
    delete[] p;
    system("pause");} ;
```

输出的结果，也令人满意：

确实调用了派生类的析构函数，而非基类的析构函数。

即使如此，是否能安心的使用一个基类指针指向派生类数组？我不太安心！——对于基类的析构 函数是否为虚函数没有把握。所以最好还是不要把一个基类的指针指向派生类数组。非得这么 做？那么我认为 `delete` 的时候将之类类型转换为派生类就差不多了，可以这样：

```
delete[] static_cast<point3d*>(p);
```

似乎不必要像 Lippman 说的这样：

```
for ( int ix = 0; ix < elem_count; ++ix ) {
    Point3d *p = &((Point3d*)ptr)[ ix ];
    delete p; }
```

参考：Lippman 的两本书《深度探索 C++对象模型》和《C++ Primer》。