C++中内存泄漏的几种情况

1. 在类的构造函数和析构函数中没有匹配的调用new和delete函数

两种情况下会出现这种内存泄露：一是在堆里创建了对象占用了内存，但是没有显示地释放对象占用的内存；二是在类的构造函数中动态的分配了内存，但是在析构函数中没有释放内存或者没有正确的释放内存

2. 没有正确地清除嵌套的对象指针

3. 在释放对象数组时在delete中没有使用方括号

方括号是告诉编译器这个指针指向的是一个对象数组，同时也告诉编译器正确的对象地址值并调用对象的析构函数，如果没有方括号，那么这个指针就被默认为只指向一个对象，对象数组中的其他对象的析构函数就不会被调用，结果造成了内存泄露。如果在方括号中间放了一个比对象数组大小还大的数字，那么编译器就会调用无效对象（内存溢出）的析构函数，会造成堆的奔溃。如果方括号中间的数字值比对象数组的大小小的话，编译器就不能调用足够多个析构函数，结果会造成内存泄露。

释放单个对象、单个基本数据类型的变量或者是基本数据类型的数组不需要大小参数，释放定义了析构函数的对象数组才需要大小参数。

4. 指向对象的指针数组不等同于对象数组

对象数组是指：数组中存放的是对象，只需要delete []p，即可调用对象数组中的每个对象的析构函数释放空间

指向对象的指针数组是指：数组中存放的是指向对象的指针，不仅要释放每个对象的空间，还要释放每个指针的空间，delete []p只是释放了每个指针，但是并没有释放对象的空间，正确的做法，是通过一个循环，将每个对象释放了，然后再把指针释放了。

5. 缺少拷贝构造函数

两次释放相同的内存是一种错误的做法，同时可能会造成堆的奔溃。

按值传递会调用（拷贝）构造函数，引用传递不会调用。

在C++中，如果没有定义拷贝构造函数，那么编译器就会调用默认的拷贝构造函数，会逐个成员拷贝的方式来复制数据成员，如果是以逐个成员拷贝的方式来复制指针被定义为将一个变量的地址赋给另一个变量。这种隐式的指针复制结果就是两个对象拥有指向同一个动态分配的内存空间的指针。当释放第一个对象的时候，它的析构函数就会释放与该对象有关的动态分配的内存空间。而释放第二个对象的时候，它的析构函数会释放相同的内存，这样是错误的。

所以，如果一个类里面有指针成员变量，要么必须显示的写拷贝构造函数和重载赋值运算符，要么禁用拷贝构造函数和重载赋值运算符

C++中构造函数，拷贝构造函数和赋值函数的区别和实现参见：http://www.cnblogs.com/liushui-sky/p/7728902.html

6. 缺少重载赋值运算符

这种问题跟上述问题类似，也是逐个成员拷贝的方式复制对象，如果这个类的大小是可变的，那么结果就是造成内存泄露，如下图:

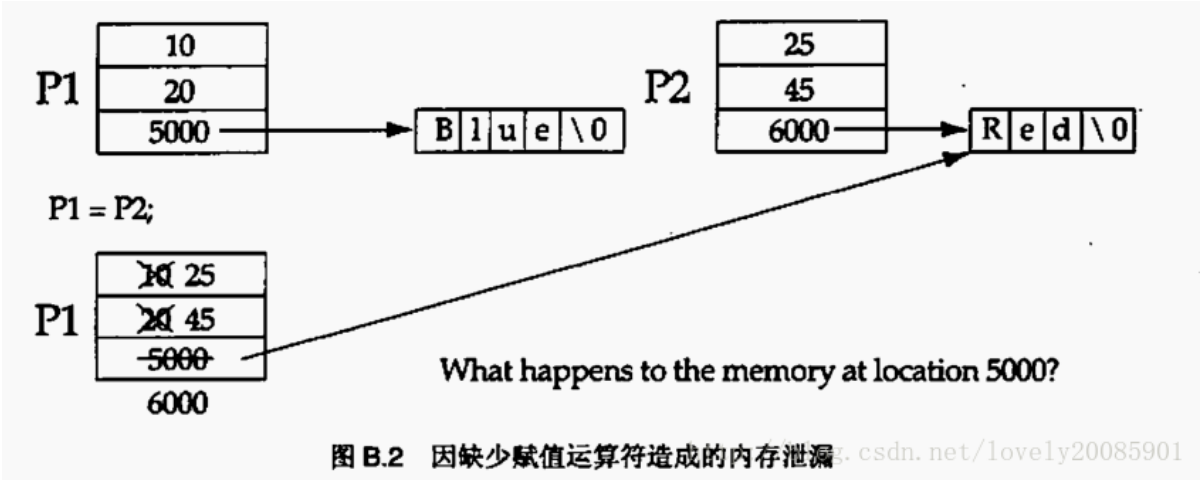

图 B.2 因缺少赋值运算符造成的内存泄漏

7. 关于nonmodifying运算符重载的常见迷思

a. 返回栈上对象的引用或者指针（也即返回局部对象的引用或者指针）。导致最后返回的是一个空引用或者空指针，因此变成野指针

b. 返回内部静态对象的引用。

c. 返回一个泄露内存的动态分配的对象。导致内存泄露，并且无法回收

解决这一类问题的办法是重载运算符函数的返回值不是类型的引用，二应该是类型的返回值，即不是 int&而是int

8. 没有将基类的析构函数定义为虚函数

当基类指针指向子类对象时，如果基类的析构函数不是virtual，那么子类的析构函数将不会被调用，子类的资源没有正确是释放，因此造成内存泄露

野指针：指向被释放的或者访问受限内存的指针。

造成野指针的原因：

1. 指针变量没有被初始化（如果值不定，可以初始化为NULL）
2. 指针被free或者delete后，没有置为NULL, free和delete只是把指针所指向的内存给释放掉，并没有把指针本身干掉，此时指针指向的是"垃圾"内存。释放后的指针应该被置为NULL.
3. 指针操作超越了变量的作用范围，比如返回指向栈内存的指针就是野指针。