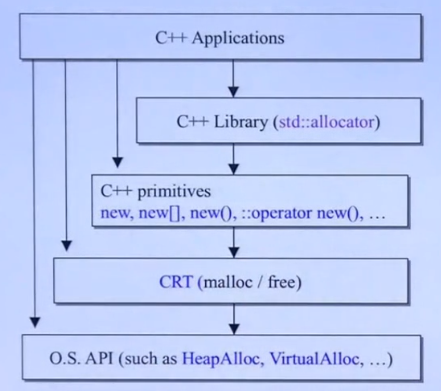
# 概述

书籍：《STL源码剖析》、《Small Memory Software》、《Modern C++ Design》

Library：STL Allocators、MFC CPlex+ CFixedAlloc、Boost.Pool、Loki SmallObjAllocator、VC malloc/free、jemalloc、tcmalloc

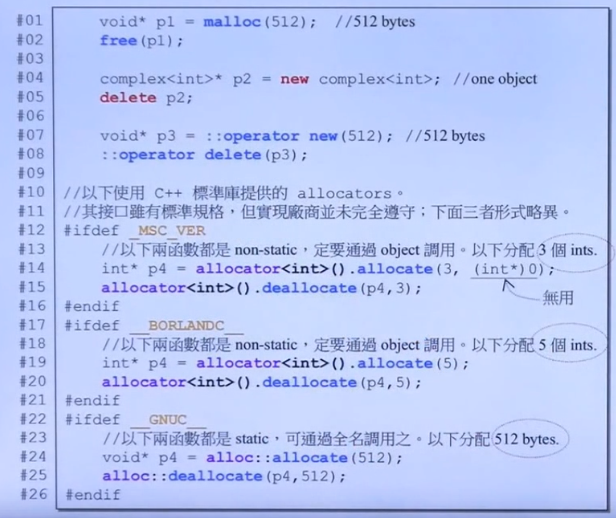
## 内存分配层次



|  |  |  |  |
| --- | --- | --- | --- |
| 分配 | 释放 | 类别 | 可否重载 |
| malloc() | free() | C函数 | 不可以 |
| new | delete | C++表达式 | 不可以 |
| ::operator new() | ::operator delete() | C++函数 | 可以 |
| allocator<T>::allocatoe() | allocator<T>::deallocate | C++标准库 | 可自由设计并搭配任何容器 |

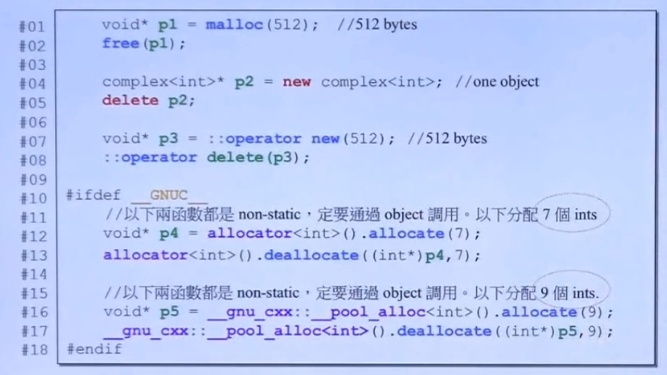
注：其实::operator new()底层调用malloc()，::operator delete()底层调用free()。

## 源码



注：allocator根据不同平台编译器：MSC、BorlandC、GUNC，执行不同的操作（比如Borland指定数据类型，GNUC不用指定，分配的是字节数）。

新版本：

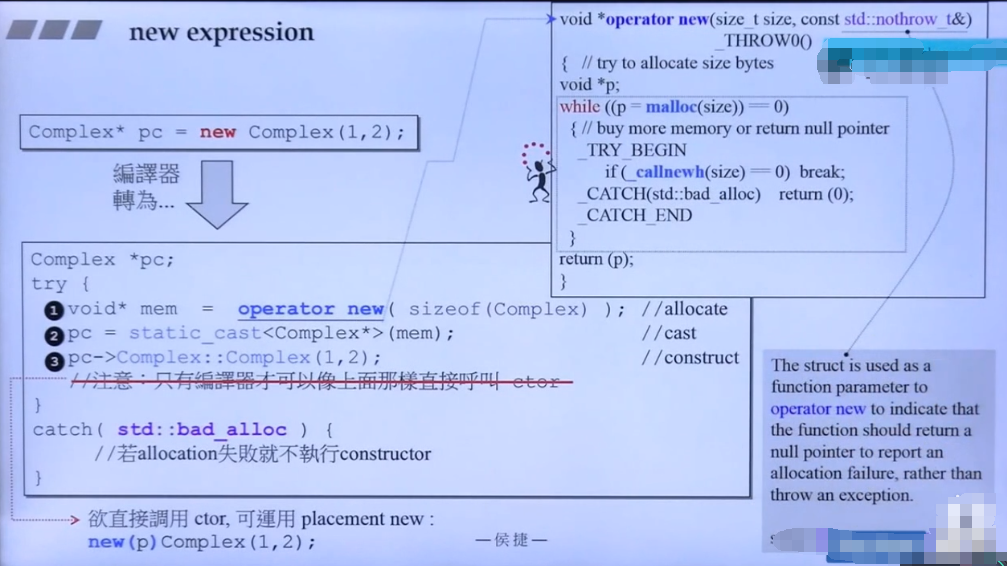


注：对于GNUC用法与BorlandC非常相似。

# new/delete

## new

new有两个作用：分配内存，自动调用构造函数。



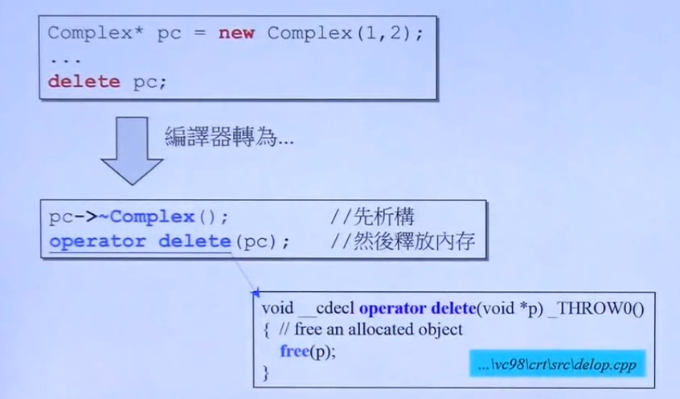
编译器转换为3步骤：

1. new调用operator new函数分配内存（可以重载，底层调用malloc），返回void\*；
2. 将分配器返回void\*类型转换为我们需要的类型；
3. 通过pc指针调用构造函数初始化。

在operator new中（VC平台），如果内存不足，调用callnewh，new handler是一种你可以自定义的函数，如果malloc失败，就可以调用new handler，可以在这个new hanlder里释放一部分无用的内存，这样就可以在内存不足的时候可以释放一些不用的内存。

## delete

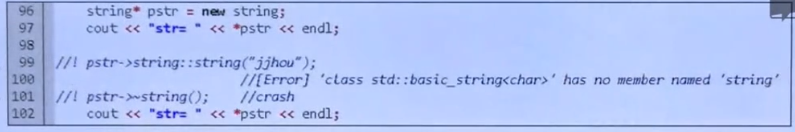
delete作用有两个：先调用析构函数，然后释放对象的内存。



注：不能直接调用构造函数，但是可以直接调用析构函数。之所以不能直接调用构造函数，是因为一开始我们并不知道具体使用哪个子类的构造方法。需要经过虚函数表找到对应的子类，调用子类的构造方法后才可以。析构的时候，由于构造的时候已经确定好了父类子类的变量和关系，所以它可以自己找到对应释放的子类析构函数。

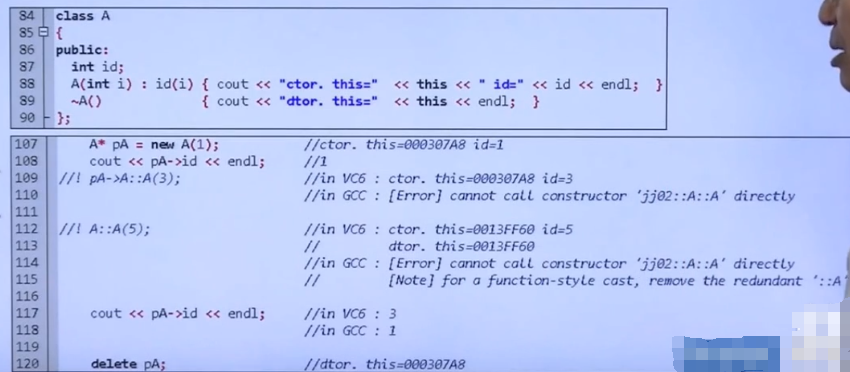
## 构造/析构函数

构造函数和析构函数直接调用：

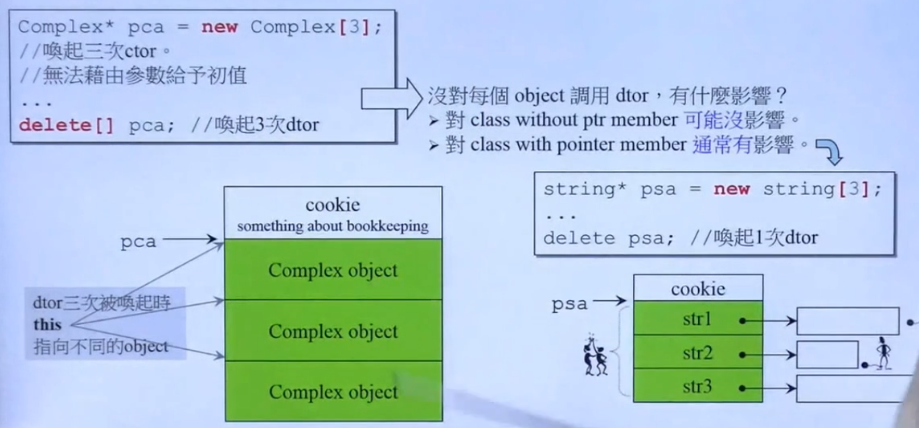


注：string在标准库里实际上是typedef类型，实际上是basic\_string，编译器在编译的时候将string转换为basic\_string，在调用string()方法的时候找不到basic\_string的这个方法，这样看报错是很合理的。

下面的析构函数调用是可以编译通过的，但是如果构造函数失败了，再调用这个析构函数就会crash（构造与析构不一一对应）。

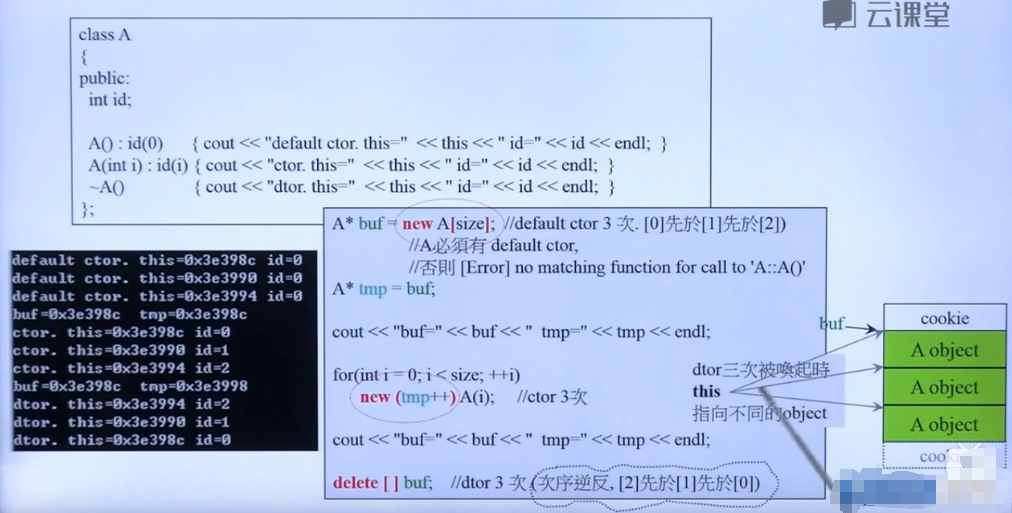


## array new



调用new[]之后，会在内存起始位置存在一个cookie（所有平台和编译器都是这样设计的），可以记录分配内存数组的长度（malloc的时候参数带长度，free的时候参数只有一个地址，它通过这里暂存的长度信息确定释放多少内存）。

注：delete的时候如果不添加[]，只会调用一次析构函数，则可能会存在内存泄露（[]的作用就是告诉编译器这是一个数组，需要释放多次）。



## array delete

# malloc/free

# std::allocator

# 其他allocator

# loki::allocator