

# 重载::operator new 和::operator delete

## 第五周作业 (NILYANG)

OS 环境: 64 位 windows 10

### 一、添加构造和析构

#### 1.1 观察结果:

```
void Test_Fruit()
{
    std::cout << "----call Test_Fruit()\n";
    pApple:
    Fruit::Fruit's ctor, this's address = 000001FF1BD44D20
    Apple::Apple's ctor , this's address = 000001FF1BD44D20
    apple:
    Fruit::Fruit's ctor, this's address = 00000089FF8FF8D8
    Apple::Apple's ctor , this's address = 00000089FF8FF8D8
    delete pApple:
    Apple::Apple's dtor
    Fruit::~Fruit's dtor
    ----exit Test_Fruit():
    Apple::~Apple's dtor
    Fruit::~Fruit's dtor
    请按任意键继续... . . .
```

上面的图片里，打印的是 `this` 指针的值，不难发现如下几个特点：

1. `Apple` 对象的构造过程是，先构造父类对象 `Fruit`，然后才是 `Apple` 自己。析构的过程是先析构 `Fruit` 然后才是 `Apple` 自己。
  - a. 印证了继承关系下，构造函数调用顺序是，由内及外，析构恰好相反，由外及里。
2. `Apple` 对象的基类部分和 `Apple` 自己的 `this` 指针所指地址是一样的。
3. 自动变量（local）分配的变量，在退出函数后才销毁，`new` 的对象，手动销毁时即销毁。
4. 无论栈空间对象还是对内存对象，`Apple` 的 `this` 和父类 `Fruit` 的 `this` 值相同。
5. 地址空间是由小到大增长，即，对象开始时是小地址，分配的对象内部数据成员是向上的，如[1.2 小节]所示：

#### 1.2 地址增长示意图（表格）:

以 `pAddress` 分配的内存空间为例，推算空间：

`sizeof(*pAddress) = 40`

名称	值
<code>sizeof(pApple)</code>	8
<code>sizeof(*pApple)</code>	40

```

---call Test_Fruit():
pApple:
Fruit::Fruit's ctor, this's address = 0000022187A642D0
Apple::Apple's ctor , this's address = 0000022187A642D0
no's address = 0000022187A642D8
weight's address = 0000022187A642E0
key's address = F屯屯屯屯
size's address = 0000022187A642F0
type's address = A屯妄
apple:
Fruit::Fruit's ctor, this's address = 0000005C0BBFF8E8
Apple::Apple's ctor , this's address = 0000005C0BBFF8E8
delete pApple:
Apple::~Apple's dtor
Fruit::~Fruit's dtor
---exit Test_Fruit():
Apple::~Apple's dtor
Fruit::~Fruit's dtor
请按任意键继续. . .

```

从下自上排列：

位	地址	名字/类型	占用空间	说明
高	未打印出地址	&type / char	4 字节	Key 是 char, 但不知其地址
	0x0000022187A642F0	&size / int	4 字节	Apple 对象成员开始
	未打印出地址	__vptr	总计 16 字节, 猜 测均分 8 字节	Fruit 虚函数指针
	未打印出地址	&key /char		Fruit 字符成员 Key
	0x0000022187A642E0	&weight / double	8 字节	
	0x0000022187A642D8	&no / int	8 字节	
低	0x0000022187A642D0	this 对象值	对象占 40 字节	Apple/Base 起始地址

## 二、自定义 ::operator new && ::operator delete

如图所示，仍然以堆分配方式的 pApple 为例，我在该类中，重载了全局的 new 和 delete 操作，然后结果分析如下：

```
---call Test_Fruit():
pApple:
fuit malloc size = 40, address = 000001960FB269C0
Fruit::Fruit's ctor, this's address = 000001960FB269C0
Apple::Apple's ctor , this's address = 000001960FB269C0
no's address = 000001960FB269C8
weight's address = 000001960FB269D0
key's address = F屯屯屯屯
size's address = 000001960FB269E0
type's address = A屯妄
delete pApple:
Apple::~Apple's dtor
Fruit::~Fruit's dtor
fuit free
delete address = 000001960FB269C0
---exit Test_Fruit():
请按任意键继续. . .
```

上图所示，

- 1. new 操作在这里生效了，制定为我自己定义的 new 操作符。
- 2.delete 操作和上面一样，也是调用了我自己定义的。
- 3.通过自定义的操作符，我们可以知道，
  - 3.1 通过 malloc 分配了 40 个字节的空间，且地址，与 this 是一致的。
  - 3.2 调用完成之后，free 的地址和 new 分配的也是一致的。
- 4.顺序
  - 4.1 分配地址操作出现的时机：
    - ◆ 在 new 操作符之后，构造函数（含父类或者复合对象）之前
  - 4.2 释放空间操作的时机：
    - ◆ 在 delete 操作符之后，对象所有（含父类，或者复合对象）析构函数之后，
    - ◆ 与 new 配对出现