老生常谈，正确使用memset

分类： 又爱又恨的C++（5）

前段项目中发现一个问题，程序总是在某个dynamic_cast进行动态转换时出异常，查了半天才发现问题原来是出在memset的使用上，虽然问题本身显而易见，但当处于几十万行代码量级中时，就变得不太那么容易定位了。

本文归纳了下使用memset几个需要注意的地方，虽然内容很简单，但也希望对大家有所帮助。

1. memset是以字节为单位，初始化内存块。

当初始化一个字节单位的数组时，可以用memset把每个数组单元初始化成任何你想要的值，比如，

```cpp
char data[10];
memset(data, 1, sizeof(data));    // right
memset(data, 0, sizeof(data));    // right
```

而在初始化其他基础类型时，则需要注意，比如,

```cpp
int data[10];
memset(data, 0, sizeof(data));    // right
memset(data, -1, sizeof(data));    // right
memset(data, 1, sizeof(data));    // wrong, data[x] would be 0x0101 instead of 1
```

2. 当结构体类型中包含指针时，在使用memset初始化时需要小心。

比如如下代码中，

```cpp
struct Parameters {
          int x;
          int* p_x;
};
Parameters par;
par.p_x = new int[10];
memset(&par, 0, sizeof(par));
```

当memset初始化时，并不会初始化p_x指向的int数组单元的值，而会把已经分配过内存的p_x指针本身设置为0，造成内存泄漏。同理，对std::vector等数据类型，显而易见也是不应该使用memset来初始化的。

3. 当结构体或类的本身或其基类中存在虚函数时，也需要谨慎使用memset。

这个问题就是在开头项目中发现的问题，如下代码中，

```cpp
class BaseParameters
{
public:
    virtual void reset() {}
};

class MyParameters : public BaseParameters
{
public:
    int data[3];
    int buf[3];
};

MyParameters my_pars;
memset(&my_pars, 0, sizeof(my_pars));
BaseParameters* pars = &my_pars;

//......

MyParameters* my = dynamic_cast<MyParameters*>(pars);
```

程序运行到dynamic_cast时发生异常。原因其实也很容易发现，我们的目的是为了初始化数据结构MyParameters里的data和buf，正常来说需要初始化的内存空间是sizeof(int) * 3 * 2 = 24字节，但是使用memset直接初始化MyParameters类型的数据结构时，sizeof(my_pars)却是28字节，因为为了实现多态机制，C++对有虚函数的对象会包含一个指向虚函数表(V-Table)的指针，当使用memset时，会把该虚函数表的指针也初始化为0，而dynamic_cast也使用RTTI技术，运行时会使用到V-Table，可此时由于与V-Table的链接已经被破坏，导致程序发生异常。