[C++类的大小——sizeof()](http://blog.csdn.net/pizi0475/article/details/6292556)

先看这么个问题——已知：

class CBase  
{  
int  a;  
char \*p;  
};

那么运行cout<<"sizeof(CBase)="<<sizeof(CBase)<<endl;之后输出什么？

这个应该很简单，两个成员变量所占的大小有嘛——8。可由时候人就是爱犯这个错误：这么简单的问题人家会问你？再想想……好像C++类里面有个什么 函数指针，也应该占字节吧！？什么指针来着？忘了（还是水平低不扎实）！流汗中……算了姑且认为是构造函数和析构函数吧。一人一个加上刚才那8个16个。 好笑吗？这是我犯的错误！！！到底C++类的sizeof是多少呢？没有所谓的函数指针问题吗？不甘心，编个例子看看：

第一步：给丫来个空的（不好意思上火粗鲁了）

class CBase  
{  
};

运行cout<<"sizeof(CBase)="<<sizeof(CBase)<<endl;

sizeof(CBase)=1；

为什么空的什么都没有是1呢？查资料……查啊查……OK这里了：先了解一个概念：类的实例化，所谓类的实例化就是在内存中分配一块地址，每个实例在 内存中都有独一无二的地址。同样空类也会被实例化（别拿豆包不当干粮，空类也是类啊），所以编译器会给空类隐含的添加一个字节，这样空类实例化之后就有了 独一无二的地址了。所以空类的sizeof为1。继续下一步：

第二步：

还是最初的那个类，运行结果：sizeof(CBase)=8

没什么说的，两个内部变量的大小。难道我记错了没有什么指针问题的存在？再试试（早这么有求知欲也不会丢人了，这回来劲了）

第三步：添个虚函数

class CBase  
{  
public:  
CBase(void);  
virtual ~CBase(void);  
private:  
int   a;  
char \*p;  
};

再运行：sizeof(CBase)=12

嗨！问题出来了！！跟虚函数有关。为什么呢？查资料ing……

有了：“C++ 类中有虚函数的时候有一个指向虚函数的指针（vptr），在32位系统分配指针大小为4字节”噢原来如此害死我了。那么继承类呢？

第四步：

基类就是上面的了不写了

class CChild :  
public CBase  
{  
public:  
CChild(void);  
~CChild(void);  
private:  
int b;  
};

运行：cout<<"sizeof(CChild)="<<sizeof(CChild)<<endl;

输出：sizeof(CChild)=16；

可见子类的大小是本身成员变量的大小加上子类的大小。

关于虚拟继承（相当于添加了一个接口）：

class COneMember

{

public:

    COneMember(int iValue = 0){m\_iOne = iValue;};

private:

    int m\_iOne;

};

class CTwoMember:virtual public COneMember

{

private:

    int m\_iTwo;

};

长度：12

内存结构：

|  |
| --- |
| E8 2F 42 00 //指针，指向一个关于偏移量的数组，且称之**虚基类偏移量表指针**  CC CC CC CC // m\_iTwo  00 00 00 00 // m\_iOne（虚基类数据成员） |

评注：virtual让长度增加了4，其实是多了一个指针，关于这个指针，确实有些复杂，别的文章有具体分析，这里就不岔开具体讲了，可认为它指向一个关于虚基类偏移量的数组，偏移量是关于虚基类数据成员的偏移量。  
关于闭合继承：

class ClassA

{

public:

    ClassA(int iValue=1){m\_iA = iValue;};

private:

    int m\_iA;

};

class ClassB：public ClassA

{

public:

    ClassB(int iValue=2){m\_iB = iValue;};

private:

    int m\_iB;

};

class ClassC： public ClassC

{

public:

    ClassC(int iValue=3){m\_iC = iValue;};

private:

    int m\_iC;

};

class CComplex :public ClassB, public ClassC

{

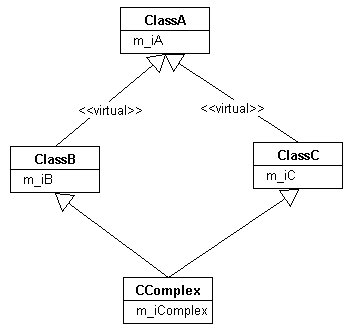
public:

    CComplex(int iValue=4){m\_iComplex = iValue;};

private:

    int m\_iComplex;

};



长度：24

内存结构：

|  |
| --- |
| 14 30 42 00 //ClassB的虚基类偏移量表指针  02 00 00 00 //m\_iB  C4 2F 42 00 //ClassC的虚基类偏移量表指针  03 00 00 00 //m\_iC  04 00 00 00 //m\_iComplex  01 00 00 00 //m\_iA |

评注：和预料中的一样，虚基类的成员m\_iA只出现了一次，而且是在最后边。当然了，更复杂的情况要比这个难分析得多，但虚继承不是我们研究的重点，我们只需要知道：虚继承利用一个“虚基类偏移量表指针”来使得虚基类即使被重复继承也只会出现一次。  
看一下关于static成员

|  |
| --- |
| class CStaticNull  {  public:      CStaticNull(){printf("Construct/n");}      ~CStaticNull(){printf("Desctruct/n");}      static void Foo(){printf("Foo/n");}      static int m\_iValue;  }; |

长度：1

内存结构：（同CNull2）

评注：可见static成员不会占用类的大小，static成员的存在区域为静态区，可认为它们是“全局”的，只是不提供全局的访问而已，这跟C的static其实没什么区别。  
带有虚函数的类：

class CVirtualNull

{

public:

    CVirtualNull(){printf("Construct/n");}

    ~CVirtualNull(){printf("Desctruct/n");}

    virtual void Foo(){printf("Foo/n");} //或者是virtual void Foo()=0;都是占4个字节大小

};

长度：4

内存结构：

|  |
| --- |
| 00 31 42 00 //指向虚函数表的指针（虚函数表后面简称“虚表”）    00423100:（虚表）  41 10 40 00 //指向虚函数Foo的指针    00401041:  E9 78 02 00 00 E9 C3 03 … //函数Foo的内容（看不懂） |

评注：带虚函数的类长度就增加了4，这个4其实就是个指针，指向虚函数表的指针，上面这个例子中虚表只有一个函数指针，值就是“0x00401041”，指向的这个地址就是函数的入口了。

子类有新的虚函数：

|  |
| --- |
| class CVirtualDerived: public CVirtualNull  {  public:      CVirtualDerived(){m\_iVD=0xFF;};      ~CVirtualDerived(){};  **virtual void Foo2(){printf("Foo2/n");};**  private:      int m\_iVD;  }; |

长度：8

内存结构：

|  |
| --- |
| 24 61 42 00 //虚表指针  FF 00 00 00 //m\_iVD    00426124:（虚表）  23 10 40 00  50 10 40 00 |

评注：虚表还是只有一张，不会因为增加了新的虚函数而多出另一张来，新的虚函数的指针将添加在复制了的虚表的后面。