讀者來信問了一個問題，有點趣味，我把它整理下來。一方面說明技術相關問題，一方面讓同學知道，如何模擬或驗證心中所想。

■讀者來信

Sent: Tuesday, January 04, 2005 9:48 PM

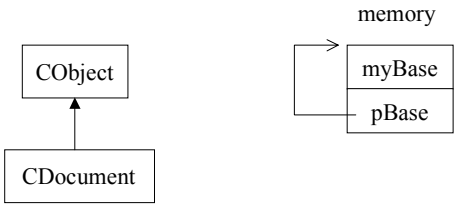
侯老師：您好！我是你忠實讀者，真的很感謝你爲我們（IT 熱愛者）作的事情！我知道侯老師很忙，但我也真的很想給你發封 E-mail 想得到你的親自指教☺ 廢話少說，是這樣的，我在看你的《深入淺出 MFC 第二版》時突然想到這個問題，直接看源程序吧，很短的。致禮！你的忠實讀者□□

```
#0001 #include <iostream>
#0002 using namespace std;
#0003
#0004 class CObject //空的基類
#0005 {};
#0006
#0007 class CDocument:public CObject  //從基類派生，注意有一個數據成員 m_data1
#0008 {                               //兩個方法只是用來讀這個數據成員而設的
#0009 public:
#0010    int m_data1;
#0011    void set() { cout<<"CDocument::set()"<<endl;
#0012                 m_data1=1; }
#0013    int get(void) const { cout<<"CDocument::get()"<<endl;
#0014                          return m_data1;}
#0015 };
#0016 //---------------------------------------
#0017 void main()
#0018 {
#0019  CObject myBase,*pBase;
#0020  pBase=&myBase;   // <== note!
#0021
#0022  ((CDocument*)pBase)->set();
#0023  //pBase 是指向 myBase 的，但強制轉換後調用 CDocument 的 set()
#0024  //對 m_data1 賦值，但 myBase 並沒有這個數據成員
#0025  //啊，那這個值放哪去了，丟了嗎，但我在下面這句裡：
#0026
#0027  int j=((CDocument*)pBase)->get();//既然可以 get 回來!
#0028
#0029  //我猜這個數據應該是存在的，而且很可能覆蓋了緊跟定義在 myBase 後面的變量，
#0030  //會造成程序無法預料的錯誤，不知道事實是不是這樣，如果是這樣，
#0031  //那緊跟 myBase 後面的變量就是 pBase 啊，爲什麼 pBase 的值沒變呢，
#0032  //或者不是這樣的? 但我覺得這樣編程肯定有問題的，請侯老師指教怎樣仿真.
#0033 }
```

■侯捷回覆

這是一個向下轉型（downcast）問題。在 C++中，向上轉型（upcast）一定安全，向下轉型（downcast）則不一定安全。

本例示意圖如下：



你問：

```
#0029   //我猜這個數據應該是存在的，而且很可能覆蓋了緊跟定義在 myBase 後面的變量，
#0030   //會造成程序無法預料的錯誤，不知道事實是不是這樣，如果是這樣，
#0031   //那緊跟 myBase 後面的變量就是 pBase 啊，爲什麼 pBase 的值沒變呢，
```

既有此疑問，就把各個位址列印出來看看 ☺

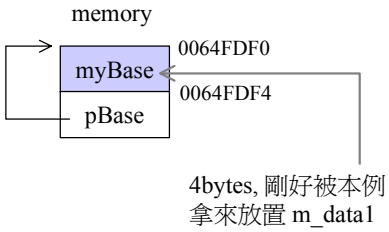
```
cout << "sizeof(myBase)=" << sizeof(myBase) << endl;
cout << "sizeof(*pBase)=" << sizeof(*pBase) << endl;
cout << "sizeof(pBase)=" << sizeof(pBase) << endl;
cout << "sizeof((CDocument*)pBase)=" << sizeof((CDocument*)pBase) << endl;
cout << "sizeof(*(CDocument*)pBase)=" << sizeof(*(CDocument*)pBase) << endl;
cout << "myBase: " << &myBase << endl;
cout << "pBase:  " << pBase << endl;
cout << "&pBase:  " << &pBase << endl;
cout << "&(*(CDocument*)pBase)=" << &(*(CDocument*)pBase) << endl;
```

獲得的結果是（**VC6**）：

```
sizeof(myBase)=1
sizeof(*pBase)=1
sizeof(pBase)=4
sizeof((CDocument*)pBase)=4
sizeof(*(CDocument*)pBase)=4
myBase: 0064FDF0
pBase:  0064FDF0
&pBase:  0064FDF4
&(*(CDocument*)pBase)=0064FDF0
```

這顯示，myBase（一個 CObject object）的大小理論上爲 0，實際上爲 1（爲了實作上的理由！這在 C++ Object Model 相關書籍中都有討論。此大小值其實取決於編譯器，例如 BCB 就和 VC 不同）。記憶體內的 myBase 的下一個變數的確是 pBase

（大小爲 4），但並非緊跟在 myBase 之後，而是在 4 倍數之後（因爲 address alignment。Address alignment 的預設值亦取決於編譯器）。



以下逐一回覆並釐清觀念：

```
#0029    //我猜這個數據應該是存在的,
```

編譯器面對 pointer 的處理方式，完全看 pointer 指向何物，也就是視 pointer 的 "dynamic type" 而定。當 pBase 被轉型爲一個 CDocument*，編譯器即把 pBase 所指位址之後的空間視爲一個 CDocument object 的佔用空間。因此本例中 m_data1 即「被視爲」存在。但它其實在不在呢？答案是，應該這麼說：被編譯器視爲 m_data1 的那塊空間，其實並不真正是 m_data1。

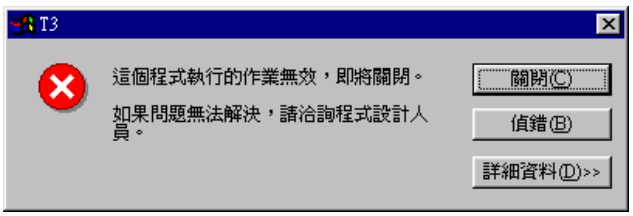
```
#0029    //我猜這個數據應該是存在的，而且很可能覆蓋了緊跟定義在 myBase 後面的變量,
#0030    //會造成程序無法預料的錯誤，不知道事實是不是這樣
```

是的。

```
#0030    //...如果是這樣,
#0031    //那緊跟 myBase 後面的變量就是 pBase 啊，爲什麼 pBase 的值沒變呢,
```

剛才說了，myBase 和 pBase 之間距離 4bytes（因 alignment 之故），這 4bytes 在本例中被當做 m_data1 剛剛好，所以陰錯陽差 OK——設置其值 OK，取其值也 OK。

如果把 CDocument 設計爲（例如）帶有一個 double m_data1（大小爲 8，超過 alignment 4），那麼對向下轉型後的 pBase 設置 m_data1 值時，便會對 pBase 帶來影響，破壞 pBase 內容，造成執行期錯誤：



在 C++ 中，爲防止 downcast 帶來的可能危險，程式中不該做 C-style 強制轉型：

```
#0022   ((CDocument*)pBase)->set();
```

必須改用 dynamic_cast 轉型運算子（編譯時需加選項 -GR）：

```
#include <cassert>
...
CDocument* pDoc = dynamic_cast<CDocument*>(pBase);
assert(pDoc);
pDoc->set();
int j = pDoc->get();
```

但是當你這麼做，編譯器會報錯：

```
error C2683: dynamic_cast : 'CObject' is not a polymorphic type
```

換句話說如果想要運用 C++ 提供之「可檢測向下轉型成功與否」的 dynamic_cast，你的 type 必須是個 polymorphic type。所謂 polymorphic type 簡單地說就是它必須具備 virtual function。

其實，任何 base class 都應該帶有一個 virtual destructor，這是個好習慣，可避免將來可能的出錯(當 derived class object 解構時)。因此我們很應該在本例的 CObject 中加上：

```
class CObject //空的基類
{
public:
   virtual ~CObject() { }
};
```

現在，重新編譯程式（cl -GX -GR test.cpp）並執行，獲得以下結果：

```
Assertion failed: pDoc, file ..\test.cpp, line 31
abnormal program termination
```

其中的 line31 便是 assert(pDoc); 這一行。這便是檢測出危險的向下轉型。

-- the end