# Why does the debugger show me the wrong virtual function?

devblogs.microsoft.com/oldnewthing/20050323-00

March 23, 2005

Raymond Chen

Pointers to virtual functions all look basically the same and therefore, as we learned last time, all end up merged into a single function. Here's a contrived example:

```
class Class1
{
public:
 virtual int f1() { return 0; }
 virtual int f2() { return 1; }
};
class Class2
{
public:
 virtual int g1() { return 2; }
 virtual int g2() { return 3; }
};
int (Class1::*pfn1)() = Class1::f2;
int (Class2::*pfn2)() = Class2::g2;
```

If you take a look at `pfn1` and `pfn2` you'll see that the point to the same function:

```
0:000> dd pfn1 l1
01002000  010010c8
0:000> dd pfn2 l1
01002004  010010c8
0:000> u 10010c8 l2
010010c8 8b01     mov     eax,[ecx]           ; first vtable
010010ca ff6004   jmp     dword ptr [eax+0x4] ; second function
```

That's because the virtual functions `Class1::f2` and `Class2::g2` are both stored in the same location relative to the respective object pointer: They are the second entry in the first vtable. Therefore, the code to call those functions is identical and consequently has been merged by the linker.

Notice that the function pointers are not direct pointers to the concrete implementations of `Class1::f2` and `Class2::g2` because the function pointer might be applied to a derived class which override the virtual function:

```
class Class3 : public Class1
{
public:
 virtual int f2() { return 9; }
};
Class3 c3;
(c3.*pfn1)(); // calls Class3::f2
```

Applying the function pointer invokes the function on the derived class, which is the whole point of declaring the function `Class1::f2` as virtual in the first place.

Note that the C++ language explicitly states that the result of comparing non-null pointers to virtual member functions is “unspecified”, which is language-standards speak for “the result not only depends on the implementation, but the implementation isn’t even required to document how it arrives at the result.”

Raymond Chen

**Follow**