Understanding warning C4265: class has virtual functions, but destructor is not virtual, part 1

devblogs.microsoft.com/oldnewthing/20200618-00

June 18, 2020

Raymond Chen

A few customers have noted that compiling WRL and C++/WinRT headers generate warning C4265: class has virtual functions, but destructor is not virtual.

What's this warning trying to say?

If a non-final class has a virtual method, then the compiler considers the possibility that there will be a derived class which overrides that virtual method. And if the class does not have a virtual destructor when a pointer to the base class is `delete`d, then only the destructor for the base class will run; the destructor for the derived class will not run.[1]

```
struct Base
{
  virtual void f();
};

struct Derived : Base
{
 ~Derived();
};
```

In the above case, the warning is generated because of this possibility:

```
void Finish(Base *base)
{
 delete base;
}

void Something()
{
  auto d = new Derived();
  Finish(d);
}
```

If a pointer points to a base class, but the pointer points at runtime to a derived class, then the base class must have a virtual destructor in order to destruct the derived class. Without the virtual destructor, only the destructor for the base class will run.

However, if you never do a `delete base`, then the problem never arises.

The compiler is warning about a potential trap, rather then waiting for you to fall into the trap. This means that if you are careful to avoid the trap, you have no actual problem, but the compiler will warn you about it anyway.[2]

If you know that nobody derives from `Derived`, you can mark it as `final`, which will make the warning go away.

Next time, we'll apply this understanding to WRL and C++/WinRT.

[1] Formally, the behavior is undefined. See **[expr.delete]** paragraph 3.

> If the static type of the object to be deleted is different from its dynamic type…, the static type shall be a base class of the dynamic type of the object to be deleted and the static type shall have a virtual destructor or the behavior is undefined.

[2] I would prefer the compiler wait until you actually do the `delete` before generating the warning. But it's being proactive and warning about a potential problem, even if that potential is not realized immediately.

Raymond Chen

Follow