

# Re: reinterpret\_cast

---

*Source:*

<http://www.tech-archive.net/Archive/Development/microsoft.public.win32.programmer.kernel/2006-04/msg00455.htm>

---

- *From:* "Slava M. Usov" <[stripit.slough@xxxxxxx](mailto:stripit.slough@xxxxxxx)>
  - *Date:* Tue, 18 Apr 2006 03:09:37 +0200
- 

"anton bassov" <[xxx@xxxxxxx](mailto:xxx@xxxxxxx)> wrote in message  
[news:c60bc5d9f40f444ba6a7a0cb83b1decc@xxxxxxxxxxxxxxxx](mailto:news:c60bc5d9f40f444ba6a7a0cb83b1decc@xxxxxxxxxxxxxxxx)

Hi Carl

Look at the statment below:

//////////

That's incorrect. struct/class makes no difference: the C-style cast still behaves like static\_cast, not reinterpret\_cast, regardless of whether the types were declared using the class keyword or the struct keyword. In fact,

in such cases the C cast is always equivalent to static\_cast (which will produce the same result as reinterpret\_cast in some cases).

//////////

This is wrong, believe me

That's not really a good way to defend your point. In this case, a quotation from the C++ standard would be much more convincing.

For example:

[begin quote "5.4 Explicit type conversion (cast notation)"]

The conversions performed by

--a const\_cast (5.2.11),

--a static\_cast (5.2.9),

--a static\_cast followed by a const\_cast,

--a reinterpret\_cast (5.2.10), or

## Re: reinterpret\_cast

—a reinterpret\_cast followed by a const\_cast,

can be performed using the cast notation of explicit type conversion. The same semantic restrictions and behaviors apply. If a conversion can be interpreted in more than one of the ways listed above, the interpretation that appears first in the list is used, even if a cast resulting from that interpretation is ill-formed.

[...]

In addition to those conversions, the following static\_cast and reinterpret\_cast operations (optionally followed by a const\_cast operation) may be performed using the cast notation of explicit type conversion, even if the base class type is not accessible:

—a pointer to an object of derived class type or an lvalue of derived class type may be explicitly converted to a pointer or reference to an unambiguous base class type, respectively;

[end quote]

The quotation above, however, means that Carl is right and you are wrong.

– the only reason why I mentioned it is because I actually tried to change struct to class in the example that you have presented. When I compiled it with struct, everything went the way you have said. However, when I changed it to class, I got the compile error, telling me that "static cast exists, but is unavailable".

This is only because that when you write

```
struct D: A, B { }
```

it really means

```
struct D: public A, public B { }
```

If you change this to

```
struct D: private A, private B { }
```

you will have exactly the same error message, so the keyword 'class' is not relevant at all.

There was no problem with reinterpret\_cast.

Therefore, I removed static\_cast, and tried to run the program – C-style cast and reinterpret\_cast worked the same way, and both returned wrong results(they both returned a pointer to class A, rather than to class B)

This means your compiler is buggy. If you had a definition like

```
class D: A, B {};
```

```
D d, *p = &d;
```

```
B *b = (B*)p;
```

the explicit cast notation should give you a valid pointer to the B base, even though it is not accessible, as remarked in the quotation above. A compiler giving you a pointer to something else is broken.

S

.