**cv (const and volatile) type qualifiers**

From cppreference.com

Appear in any type specifier, including *decl-specifier-seq* of [declaration grammar](http://en.cppreference.com/w/cpp/language/declarations), to specify constness or volatility of the object being declared or of the type being named.

* **const** - defines that the type is *constant*.
* **volatile** - defines that the type is *volatile*.
* **mutable** - applies to non-static [class members](http://en.cppreference.com/w/cpp/language/data_members) of non-reference non-const type and specifies that the member does not affect the externally visible state of the class (as often used for mutexes, memo caches, lazy evaluation, and access instrumentation). mutable members of *const* class instances are modifiable. (Note: the C++ language grammar treats mutable as a [storage-class-specifier](http://en.cppreference.com/w/cpp/language/storage_duration), but it does not affect storage class.)

|  |
| --- |
|  |

**[**[**edit**](http://en.cppreference.com/mwiki/index.php?title=cpp/language/cv&action=edit&section=1)**]Explanation**

For any type T (including incomplete types), other than [function type](http://en.cppreference.com/w/cpp/language/functions) or [reference type](http://en.cppreference.com/w/cpp/language/reference), there are three more distinct types in the C++ type system: *const-qualified* T, *volatile-qualified* T, and *const-volatile-qualified* T.

Note: [array types](http://en.cppreference.com/w/cpp/language/array) are considered to have the same cv-qualification as their element types.

When an object is first created, the cv-qualifiers used (which could be part of *decl-specifier-seq* or part of a *declarator* in a [declaration](http://en.cppreference.com/w/cpp/language/declarations), or part of *type-id* in a [new-expression](http://en.cppreference.com/w/cpp/language/new)) determine the constness or volatility of the object, as follows:

* ***const object*** - an object whose type is *const-qualified*, or a non-mutable subobject of a const object. Such object cannot be modified: attempt to do so directly is a compile-time error, and attempt to do so indirectly (e.g., by modifying the const object through a reference or pointer to non-const type) results in undefined behavior.
* ***volatile object*** - an object whose type is *volatile-qualified*, or a subobject of a volatile object, or a mutable subobject of a const-volatile object. Every access (read or write operation, member function call, etc.) made through a glvalue expression of volatile-qualified type is treated as a visible side-effect for the [purposes of optimization](http://en.cppreference.com/w/cpp/language/as_if) (that is, within a single thread of execution, volatile accesses cannot be optimized out or reordered with another visible side effect that is [sequenced-before](http://en.cppreference.com/w/cpp/language/eval_order) or sequenced-after the volatile access. This makes volatile objects suitable for communication with a [signal handler](http://en.cppreference.com/w/cpp/utility/program/signal), but not with another thread of execution, see [std::memory\_order](http://en.cppreference.com/w/cpp/atomic/memory_order)). Any attempt to refer to a volatile object through a non-volatile [glvalue](http://en.cppreference.com/w/cpp/language/value_category#glvalue) (e.g. through a reference or pointer to non-volatile type) results in undefined behavior.
* ***const volatile object*** - an object whose type is *const-volatile-qualified*, a non-mutable subobject of a const volatile object, a const subobject of a volatile object, or a non-mutable volatile subobject of a const object. Behaves as both a const object and as a volatile object.

|  |  |
| --- | --- |
|  | This section is incomplete Reason: should discuss more about the differences between cv-qualified objects and cv-qualified expressions |

There is partial ordering of cv-qualifiers by the order of increasing restrictions. The type can be said *more* or *less* cv-qualified then:

* *unqualified* < const
* *unqualified* < volatile
* *unqualified* < const volatile
* const < const volatile
* volatile < const volatile

References and pointers to cv-qualified types may be [implicitly converted](http://en.cppreference.com/w/cpp/language/implicit_cast#Qualification_conversions) to references and pointers to *more cv-qualified* types. In particular, the following conversions are allowed:

* reference/pointer to *unqualified* type can be converted to reference/pointer to const
* reference/pointer to *unqualified* type can be converted to reference/pointer to volatile
* reference/pointer to *unqualified* type can be converted to reference/pointer to const volatile
* reference/pointer to const type can be converted to reference/pointer to const volatile
* reference/pointer to volatile type can be converted to reference/pointer to const volatile

Note: [additional restrictions](http://en.cppreference.com/w/cpp/language/implicit_cast#Qualification_conversions) are imposed on multi-level pointers.

To convert a reference or a pointer to a cv-qualified type to a reference or pointer to a *less cv-qualified* type, [const\_cast](http://en.cppreference.com/w/cpp/language/const_cast) must be used.

**[**[**edit**](http://en.cppreference.com/mwiki/index.php?title=cpp/language/cv&action=edit&section=2)**] Keywords**

[const](http://en.cppreference.com/w/cpp/keyword/const), [volatile](http://en.cppreference.com/w/cpp/keyword/volatile), [mutable](http://en.cppreference.com/w/cpp/keyword/mutable)

**[**[**edit**](http://en.cppreference.com/mwiki/index.php?title=cpp/language/cv&action=edit&section=3)**] Example**

**Run this code**

int main()

{

int n1 = 0; // non-const object

const int n2 = 0; // const object

int const n3 = 0; // const object (same as n2)

volatile int n4 = 0; // volatile object

const struct

{

int n1;

mutable int n2;

} x = {0, 0}; // const object with mutable member

n1 = 1; // ok, modifiable object

// n2 = 2; // error: non-modifiable object

n4 = 3; // ok, treated as a side-effect

// x.n1 = 4; // error: member of a const object is const

x.n2 = 4; // ok, mutable member of a const object isn't const

const int& r1 = n1; // reference to const bound to non-const object

// r1 = 2; // error: attempt to modify through reference to const

const\_cast<int&>(r1) = 2; // ok, modifies non-const object n1

const int& r2 = n2; // reference to const bound to const object

// r2 = 2; // error: attempt to modify through reference to const

// const\_cast<int&>(r2) = 2; // undefined behavior: attempt to modify const object n2

}

Output:

# typical machine code produced on an x86\_64 platform

# (only the code that contributes to observable side-effects is emitted)

main:

movl $0, -4(%rsp) # volatile int n4 = 0;

movl $3, -4(%rsp) # n4 = 3;

xorl  %eax, %eax # return 0 (implicit)

ret

**[**[**edit**](http://en.cppreference.com/mwiki/index.php?title=cpp/language/cv&action=edit&section=4)**]**