class template

std::atomic

<atomic>

```
template <class T> struct atomic;
```

Atomic

Objects of atomic types contain a value of a particular type (T).

The main characteristic of atomic objects is that access to this contained value from different threads cannot cause data races (i.e., doing that is *well-defined behavior*, with accesses properly sequenced). Generally, for all other objects, the possibility of causing a data race for accessing the same object concurrently qualifies the operation as *undefined behavior*.

Additionally, `atomic` objects have the ability to synchronize access to other non-atomic objects in their threads by specifying different *memory orders*.

Template parameters

T

Type of the contained value.
This shall be a *trivially copyable type*.

Member functions

(constructor)	Construct atomic (public member function)
operator=	Assign contained value (public member function)

General atomic operations

is_lock_free	Is lock-free (public member function)
store	Modify contained value (public member function)
load	Read contained value (public member function)
operator T	Access contained value (public member function)
exchange	Access and modify contained value (public member function)
compare_exchange_weak	Compare and exchange contained value (weak) (public member function)
compare_exchange_strong	Compare and exchange contained value (strong) (public member function)

Operations supported by certain specializations (integral and/or pointer, see below)

fetch_add	Add to contained value (public member function)
fetch_sub	Subtract from contained value (public member function)
fetch_and	Apply bitwise AND to contained value (public member function)
fetch_or	Apply bitwise OR to contained value (public member function)
fetch_xor	Apply bitwise XOR to contained value (public member function)
operator++	Increment container value (public member function)
operator--	Decrement container value (public member function)
atomic::operator (comp. assign.) Compound assignments (public member function)	

Template specializations

The `atomic` class template is fully specialized for all *fundamental integral types* (except `bool`), and any extended integral types needed for the typedefs in `<cstdint>`. These specializations have the following additional member functions:

specializations	additional member functions

`char`
`signed char`
`unsigned char`
`short`
`unsigned short`
`int`
`unsigned int`
`long`
`unsigned long`
`long long`
`unsigned long long`
`char16_t`
`char32_t`
`wchar_t`
extended integral types (if any)
	`atomic::fetch_add`
	`atomic::fetch_sub`
	`atomic::fetch_and`
	`atomic::fetch_or`
	`atomic::fetch_xor`
	`atomic::operator++`
	`atomic::operator--`
	`operator (comp. assign.)`

For `bool` instantiations, only the general atomic operations are supported.
Note that most of the *C-style atomic types* are aliases of these specializations (or aliases of a base class inherited by these specializations).

`atomic` is also partially specialized for all pointer types, with the following additional member functions:

specializations	additional member functions

`U*`
(for any type U)
	`atomic::fetch_add`
	`atomic::fetch_sub`
	`atomic::operator++`
	`atomic::operator--`
	`operator (comp. assign.)`