# Atomics

In this lesson, we will discuss atomics and the different kinds of atomic operations.

WE'LL COVER THE FOLLOWING



Atomics are the base of the C++ memory model. By default, the strong version of the memory model is applied to atomics. Therefore, it is important to understand the features of the strong memory model and their relation to atomics.

> 💡 Atomic operations on atomics define the synchronization and ordering constraints.

## Kinds of Atomic Operations #

There are three different kinds of atomic operations:

- Read operation: `memory_order_acquire` and `memory_order_consume`
- Write operation: `memory_order_release`
- Read-modify-write operation: `memory_order_acq_rel` and `memory_order_seq_cst`

`memory_order_relaxed` defines neither synchronization nor ordering constraints, and it does not fit in this taxonomy.

The following table orders the atomic operations based on their various reading and/or writing characteristics.

| Operation | read | write | read-modify- |
|---|---|---|---|

| Operation | read | write | read-modify-write |
|---|---|---|---|
| `test_and_set` | | | yes |
| `clear` | | yes | |
| `is_lock_free` | yes | | |
| `load` | yes | | |
| `store` | | yes | |
| `exchange` | | | yes |
| `compare_exchange_strong` `compare_exchange_weak` | | | yes |
| `fetch_add`, `+=` `fetch_sub`, `-=` | | | yes |
| `fetch_or`, `\|=` `fetch_and`, `&=` `fetch_xor`, `^=` | | | yes |
| `++`, `--` | | | yes |

> **Note:** There is no multiplication or division in atomic operations.

If you use an atomic operation `atomVar.load()` with a memory model that is designed for a write or read-modify-write operation, the write part has no effect. The result is that

- operation `atomVar.load(std::memory_order_acq_rel)` is equivalent to operation `atomVar.load(std::memory_order_acquire)`.

- operation `atomVar.load(std::memory_order_release)` is equivalent to `atomVar.load(std::memory_order_relaxed)`.

```
std::atomic<int> atom;

atom.load(std::memory_order_acq_rel) //== atom.load(std::memory_order_acquire)

atom.load(std::memory_order_release) //== atom.load(std::memory_order_relaxed)
```

> 🔑 Refer to concurrency course for a better understanding of atomics.

---

In the next lesson, we will further discuss synchronization and ordering.