# Conventions

What are the rules and terminologies needed to run algorithms? Let's find out.

WE'LL COVER THE FOLLOWING



To use the algorithms, we have to keep a few rules in our mind.

## Headers

The algorithms are defined in various headers:

| Names | Description |
| --- | --- |
| `<algorithm>` | Contains the general algorithms. |
| `<numeric>` | Contains numeric algorithms. |
| `<functional>` | Predefined function objects and function adapters. |

## Name Conventions

Many of the algorithms have the name suffix `_if` and `_copy`.

| Names | Description |
| --- | --- |

| | |
|---|---|
| `_if` | The algorithm can be parametrized by a predicate. |
| `_copy` | The algorithm copies its elements in another range. |

Algorithms like `auto num = std::count(InpIt first, InpIt last, const T& val)` return the number of elements that are equal to `val` in the given range `[first, last)`. `num` is of type `iterator_traits<InpIt>::difference_type`. This means we have the guarantee that `num` is sufficient to hold the result, and because of the automatic return type deduction with `auto`, the compiler will give us the right types.

> 🗝️ **If the container uses an additional range, it has to be valid**
> The algorithm std::copy_if uses an iterator for the beginning of its destination range. This destination range has to be valid.

> **i Naming conventions for the algorithms**
> We use a few naming conventions for the type of arguments and the return type of the algorithms to make them easier to read.

| Name | Description |
|---|---|
| `InIt` | [Input iterator] |
| `FwdIt` | [Forward iterator] |
| `BiIt` | [Bidirectional iterator] |
| `UnFunc` | [Unary callable] |
| `BiFunc` | [Binary callable] |

| | |
|---|---|
| `UnPre` | [Unary predicate] |
| `BiPre` | [Binary predicate] |
| `Search` | The searcher encapsulates the search algorithm. |
| `ValType` | From the input range automatically deduced value type. |
| `ExePol` | [Execution policy] |

**Signature of the algorithms**

## Further information #

- std::copy_if
- searcher

---

In the next lesson, we'll discuss the `for_each` method and how it is used.