Conventions

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

To use the algorithms, you have to keep a few rules in your head.

The algorithms are defined in various headers.

`<algorithm>`:
Contains the general algorithms.
`<numeric>`:
Contains the numeric algorithms. Many of the algorithms have the name suffix `_if` and `_copy`.
`_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`. `num` is of type `iterator_traits<InpIt>::difference_type`. You have the guarantee that `num` is sufficient to hold the result. Because of the automatic return type deduction with `auto`, the compiler will give you the right types.

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

> **i Naming conventions for the algorithms**
> I 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