Search Ranges

Need to acquire a sub-range from any existing range? std::search solves the problem efficiently.

`std::search`	Searches for a range in another range from the beginning.
`std::find_end`	Searches for a range in another range from the end.
`std::search_n`	Searches for `n` consecutive elements in the range.

All algorithms that take a forward iterator can be parametrized by a binary predicate and return an end iterator for the first range, if the search was not successful.
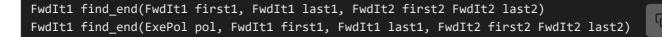
Searches the values of the second range in the first range and returns the position. Starts at the beginning:

```
FwdIt1 search(FwdIt1 first1, FwdIt1 last1, FwdIt2 first2, FwdIt2 last2)
FwdIt1 search(ExePol pol, FwdIt1 first1, FwdIt1 last1, FwdIt2 first2, FwdIt2 last2)

FwdIt1 search(FwdIt1 first1, FwdIt1 last1, FwdIt2 first2, FwdIt2 last2, BiPre pre)
FwdIt1 search(ExePol pol, FwdIt1 first1, FwdIt1 last1, FwdIt2 first2, FwdIt2 last2, BiPre pre

FwdIt1 search(FwdIt1 first, FwdIt last1, Search search)
```

Searches the values of the second range in the first range and returns the position. Starts at the end:

```
FwdIt1 find_end(FwdIt1 first1, FwdIt1 last1, FwdIt2 first2 FwdIt2 last2)
FwdIt1 find_end(ExePol pol, FwdIt1 first1, FwdIt1 last1, FwdIt2 first2 FwdIt2 last2)
```

```
FwdIt1 find_end(FwdIt1 first1, FwdIt1 last1, FwdIt2 first2, FwdIt2 last2, BiPre pre)
FwdIt1 find_end(ExePol pol, FwdIt1 first1, FwdIt1 last1, FwdIt2 first2, FwdIt2 last2, BiPre p
```

Searches `count` for consecutive values in the first range:

```
FwdIt search_n(FwdIt first, FwdIt last, Size count, const T& value)
FwdIt search_n(ExePol pol, FwdIt first, FwdIt last, Size count, const T& value)

FwdIt search_n(FwdIt first, FwdIt last, Size count, const T& value, BiPre pre)
FwdIt search_n(ExePol pol, FwdIt first, FwdIt last, Size count, const T& value, BiPre pre)
```

> ⚠ **The algorithm `search_n` is very special**
>
> The algorithm `FwdIt search_n(FwdIt first, FwdIt last, Size count, const T& value, BiPre pre)` is very special. The binary predicate `BiPre` uses, as first argument, the values of the range and, as second argument, the value `value`.

```cpp
#include <algorithm>
#include <array>
#include <cmath>
#include <iostream>

int main(){

  std::cout << std::endl;

  std::array<int, 10> arr1{0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
  std::array<int, 5> arr2{3, 4, -5, 6, 7};

  auto fwdIt= std::search(arr1.begin(), arr1.end(), arr2.begin(), arr2.end());

  if (fwdIt == arr1.end()) std::cout << "arr2 not in arr1." << std::endl;
  else{
    std::cout << "arr2 at position " << std::distance(arr1.begin(), fwdIt)    << " in arr1." <
  }

  auto fwdIt2= std::search(arr1.begin(), arr1.end(), arr2.begin(), arr2.end(), [](int a, int

  if (fwdIt2 == arr1.end()) std::cout << "arr2 not in arr1." << std::endl;
  else{
    std::cout << "arr2 at position " << std::distance(arr1.begin(), fwdIt2)   << " in arr1." <
  }

  std::cout << std::endl;

}
```

▷   💾   ↩   ⛶

std::find, std::find_end, and std::search_n

This is all from non-modifying algorithms. In the next chapter, we'll study modifying algorithms.