Merge Operation

In this lesson, we'll learn different ways of combining ranges.

Merge operations empower us to merge sorted ranges in a new sorted range. The `merge` algorithm requires that the ranges and the algorithm use the same sorting criteria. If not, the program is undefined. Per default, the predefined sorting criterion `std::less` is used. If we use our sorting criterion, it has to obey the strict weak ordering. If not, the program is undefined.

We can merge two sorted ranges with `std::inplace_merge` and `std::merge`. We can check with `std::includes` if one sorted range is in another sorted range. We can merge two sorted ranges with `std::set_difference`, `std::set_intersection`, `std::set_symmetric_difference` and `std::set_union` merge two sorted ranges in a new sorted range.

`inplace_merge`: Merges two sorted ranges `[first, mid]` and `[mid, last]` in place.

```
void inplace_merge(BiIt first, BiIt mid, BiIt last)
void inplace_merge(ExePol pol, BiIt first, BiIt mid, BiIt last)

void inplace_merge(BiIt first, BiIt mid, BiIt last, BiPre pre)
void inplace_merge(ExePol pol, BiIt first, BiIt mid, BiIt last, BiPre pre)
```

`merge`: Merges two sorted ranges and copies the result to `result`.

```
OutIt merge(InpIt first1, InpIt last1, InpIt first2, InpIt last2, OutIt result)
FwdIt3 merge(ExePol pol, FwdIt1 first1, FwdIt1 last1, FwdIt2 first2, FwdIt2 last2, FwdIt3 res

OutIt merge(InpIt first1, InpIt last1, InpIt first2, InpIt last2, OutIt result, BiPre pre)
FwdIt3 merge(ExePol pol, FwdIt1 first1, FwdIt1 last1, FwdpIt2 first2, FwdIt2 last2, FwdIt3 re
```

`includes`: Checks if all elements of the second range are in the first range.

```
bool includes(InpIt first1, InpIt last1, InpIt1 first2, InpIt1 last2)
bool includes(ExePol pol, FwdIt first1, FwdIt last1, FwdIt1 first2, FwdIt1 last2)
```

```
bool includes(InpIt first1, InpIt last1, InpIt first2, InpIt last2, BinPre pre)
bool includes(ExePol pol, FwdIt first1, FwdIt last1, FwdIt1 first2, FwdIt1 last2, BinPre pre)
```

`set_difference`: Copies the elements of the first range which are not in the second range, to `result`.

```
OutIt set_difference(InpIt first1, InpIt last1, InpIt1 first2, InpIt2 last2, OutIt result)
FwdIt2 set_difference(ExePol pol, FwdIt first1, FwdIt last1, FWdIt1 first2, FwdIt1 last2, Fwd

OutIt set_difference(InpIt first1, InpIt last1, InpIt1 first2, InpIt2 last2, OutIt result, Bi
FwdIt2 set_difference(ExePol pol, FwdIt first1, FwdIt last1, FwdIt1 first2, FwdIt1 last2, Fwd
```

`set_intersection`: Determines the intersection of the first and second ranges and copies the result to `result`.

```
OutIt set_intersection(InpIt first1, InpIt last1, InpIt1 first2, InpIt2 last2, OutIt result)
FwdIt2 set_intersection(ExePol pol, FwdIt first1, FwdIt last1, FWdIt1 first2, FwdIt1 last2, F

OutIt set_intersection(InpIt first1, InpIt last1, InpIt1 first2, InpIt2 last2, OutIt result,
FwdIt2 set_intersection(ExePol pol, FwdIt first1, FwdIt last1, FwdIt1 first2, FwdIt1 last2, F
```

`set_symmetric_difference`: Determines the symmetric difference of the first and second ranges and copies the result to `result`.

```
OutIt set_symmetric_difference(InpIt first1, InpIt last1, InpIt1 first2, InpIt2 last2, OutIt
FwdIt2 set_symmetric_difference(ExePol pol, FwdIt first1, FwdIt last1, FWdIt1 first2, FwdIt1

OutIt set_symmetric_difference(InpIt first1, InpIt last1, InpIt1 first2, InpIt2 last2, OutIt
FwdIt2 set_symmetric_difference(ExePol pol, FwdIt first1, FwdIt last1, FwdIt1 first2, FwdIt1
```

`set_union`: Determines the union of the first and second range and copies the result to `result`.

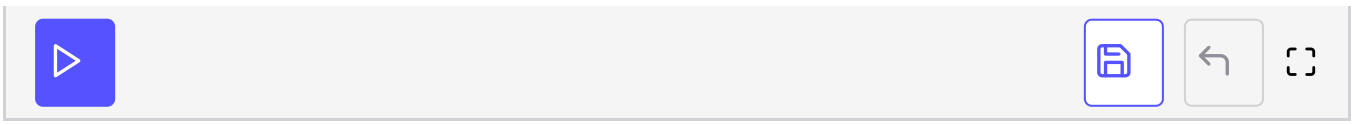
```
OutIt set_union(InpIt first1, InpIt last1, InpIt1 first2, InpIt2 last2, OutIt result)
FwdIt2 set_union(ExePol pol, FwdIt first1, FwdIt last1, FWdIt1 first2, FwdIt1 last2, FwdIt2 r

OutIt set_union(InpIt first1, InpIt last1, InpIt1 first2, InpIt2 last2, OutIt result, BiPre p
FwdIt2 set_union(ExePol pol, FwdIt first1, FwdIt last1, FwdIt1 first2, FwdIt1 last2, FwdIt2 r
```

The returned iterator is an end iterator for the destination range. The destination range of `std::set_difference` has those elements of the first range which are not found in the second range. On the contrary, the destination range of `std::symmetric_difference` has only the elements that are present in either of the ranges, but not both. `std::union` determines the union of both

sorted ranges.

```
#include <algorithm>
#include <deque>
#include <iostream>
#include <iterator>
#include <vector>

int main(){

  std::cout << std::boolalpha;

  std::vector<int> vec1{1, 1, 4, 3, 5, 8, 6, 7, 9, 2};
  std::vector<int> vec2{1, 2, 3};

  std::cout << "vec1:\t\t\t\t\t";
  for (auto v: vec1) std::cout << v << " ";
  std::cout << std::endl;
  //vec1:       1 1 4 3 5 8 6 7 9 2
  std::cout << "vec2:\t\t\t\t\t";
  for (auto v: vec2) std::cout << v << " ";
  std::cout << std::endl;
  //vec2:       1 2 3

  std::sort(vec1.begin(), vec1.end());
  std::vector<int> vec3(vec1);

  std::cout << std::endl;
  std::cout << "vec1 includes vec2: " << std::includes(vec1.begin(), vec1.end(), vec2.begin()
  //vec1 includes vec2: true
  std::cout << std::endl;

  vec1.reserve(vec1.size() + vec2.size());
  vec1.insert(vec1.end(), vec2.begin(), vec2.end());

  std::cout << "vec1:\t\t\t\t\t";
  for (auto v: vec1) std::cout << v << " ";
  std::cout << std::endl;

  std::inplace_merge(vec1.begin(), vec1.end() - vec2.size(), vec1.end());
  std::cout << "vec1:\t\t\t\t\t";
  for ( auto v: vec1 ) std::cout << v << " ";

  std::cout << "\n\n";

  vec2.push_back(10);

  std::cout << "vec3:\t\t\t\t\t";
  for (auto v: vec3) std::cout <<  v << " ";
  std::cout << std::endl;
  std::cout << "vec2:\t\t\t\t\t";
  for (auto v: vec2) std::cout << v << " ";

  std::vector<int> res;
  std::set_union(vec3.begin(), vec3.end(), vec2.begin(), vec2.end(),
         std::back_inserter(res));
  std::cout << "\n" << "set_union:\t\t\t\t";
  for (auto v : res) std::cout << v << " ";

  res={};
```

```
  std::set_intersection(vec3.begin(), vec3.end(), vec2.begin(), vec2.end(),
         std::back_inserter(res));
  std::cout << "\n" << "set_intersection:\t\t\t";

  for (auto v : res) std::cout << v << " ";


  res={};
  std::set_difference(vec3.begin(), vec3.end(), vec2.begin(), vec2.end(),
         std::back_inserter(res));
  std::cout << "\n" << "set_difference:\t\t\t";
  for (auto v : res) std::cout << v << " ";

  res={};
  std::set_symmetric_difference(vec3.begin(), vec3.end(), vec2.begin(), vec2.end(),
         std::back_inserter(res));
  std::cout << "\n" << "set_symmetric_difference:\t";
  for (auto v : res) std::cout << v << " ";


  std::cout << "\n\n";

}
```



Merge algorithms

In the next lesson, we'll learn about heaps.