Range Minimum Query

Dada una lista, permite preprocesarla para poder contestar preguntas de tipo "¿Cuál es el índice con el valor mínimo en el rango [L,R)?"

- Tiempo de preprocesamiento: $O(n \log(n))$
- Tiempo para contestar pregunta: $O(1)$.

Permite definir qué significa "menor qué".

```
#include <algorithm>
#include <cassert>
#include <cmath>
#include <iostream>
#include <numeric>
#include <vector>

template <typename RAContainer,
          typename Compare = std::less<typename RAContainer::value_type>>
class range_min_query
{
    using index_type = std::make_signed_t<size_t>;
    using Row = std::vector<index_type>;
    using value_type = typename RAContainer::value_type;

public:
    range_min_query(const RAContainer& A,
                    Compare comp = std::less<value_type>())
        : A_(A), Table(A.size(), Row(std::log2(A.size()) + 1, -1)), comp_(comp)
    {
        index_type n = A.size();
        index_type max_h = Table[0].size();

        for (index_type x = 0; x < n; ++x)
        {
            Table[x][0] = x;
        }

        for (index_type h = 1; h < max_h; ++h)
        {
            for (index_type x = 0; x < n; ++x)
            {
                if (x + (1 << h) <= n)
                {
                    index_type mid = x + (1 << (h - 1));
                    Table[x][h] = best(Table[x][h - 1], Table[mid][h - 1]);
```

```
                }
            }
        }
    }

    // Get min index in range [L,R)
    index_type GetMinIndex(index_type L, index_type R) const
    {
        assert(0 <= L && L < R && R <= A_.size());
        index_type h = std::log2(R - L);

        index_type min_index_starting_at_L = Table[L][h];
        index_type min_index_ending_at_R = BestEndingAt(R - 1, h);

        return best(min_index_starting_at_L, min_index_ending_at_R);
    }

private:
    const RAContainer& A_;
    // Table[x][i] contains the index of the
    // minimum of range [x,x+1,...,x+2^i)
    std::vector<Row> Table;

    Compare comp_;

    index_type best(index_type i, index_type j) const
    {
        if (comp_(A_[j], A_[i]))
            return j;
        return i;
    }

    index_type BestEndingAt(index_type R, index_type h) const
    {
        return Table[R - (1 << h) + 1][h];
    }
};

// This function is deprecated with C++17, but useful in c++14 and 11
template <typename RAContainer,
          typename Compare = std::less<typename RAContainer::value_type>>
range_min_query<RAContainer, Compare> make_range_min_query(
  const RAContainer& A,
  Compare comp = std::less<typename RAContainer::value_type>())
{
```

```
    return range_min_query<RAContainer, Compare>(A, comp);
}

using namespace std;

template <class T>
std::ostream& operator<<(std::ostream& os, const std::vector<T>& A)
{
    for (const auto& x : A)
        os << x << ' ';
    return os;
}

int main()
{
    std::vector<int> A = {1, 5, 3,  9, 6, 10, 1, 5, 7, 9, 8, 0, 7,
                          4, 2, 10, 2, 3, 8,  6, 5, 7, 8, 9, 9};
    auto RMQ = make_range_min_query(A);

    cout << "A = " << A << endl;
    cout << "Min value between index 5 and index "
            "15 is at: "
         << RMQ.GetMinIndex(5, 15) << " with val " << A[RMQ.GetMinIndex(5, 15)]
         << std::endl;
    auto GRMQ = make_range_min_query(A, std::greater<>());
    cout << "And the max value is at: " << GRMQ.GetMinIndex(5, 15)
         << " with val " << A[GRMQ.GetMinIndex(5, 15)] << endl;
}
```

Output:

```
A = 1 5 3 9 6 10 1 5 7 9 8 0 7 4 2 10 2 3 8 6 5 7 8 9 9
Min value between index 5 and index 15 is at: 11 with val 0
And the max value is at: 5 with val 10
```