NumberSeries

Ane Søgaard Jørgensen

February 25, 2022

Listing 1: NumberSeries.h

```h
//
// Created by ane on 18/02/2022.
//

#ifndef SESSION_5_NUMBERSERIES_H
#define SESSION_5_NUMBERSERIES_H


#include <vector>
#include <random>
#include <functional>
#include <iostream>

namespace series {

    class NumberSeries {
        std::vector<int> series{};
        int maximum{std::numeric_limits<int>::min()};
        int minimum{std::numeric_limits<int>::max()};

    public:
        int GetMaximum() {
            return maximum;
        }

        int GetMimimum() {
            return minimum;
        }

        void AddNumber(int);

        void RemoveNumbers(int);

        static NumberSeries MakeRandom(int);

        NumberSeries operator+(const NumberSeries&);

        NumberSeries& operator+=(const NumberSeries&);

        bool operator<(const NumberSeries&) const;

        friend std::ostream& operator<<(std::ostream&, const NumberSeries&);
    };

}


#endif //SESSION_5_NUMBERSERIES_H
```

Listing 2: NumberSeries.cpp

```cpp
//
// Created by ane on 18/02/2022.
//

#include <chrono>
#include "NumberSeries.h"

namespace series {

    void NumberSeries::AddNumber(int num) {
        if (num < minimum) {
            minimum = num;
        }
        else if (num > maximum) {
            maximum = num;
        }

        series.push_back(num);
    }

    void NumberSeries::RemoveNumbers(int num) {
        auto it = std::remove_if(series.begin(), series.end(), [num](int x) -> bool {
            return x == num;
        });

        series.erase(it, std::end(series));
    }

    NumberSeries NumberSeries::MakeRandom(int length) {
        NumberSeries result{};

        std::random_device dev{};
        std::default_random_engine random_engine {dev()};
        std::uniform_int_distribution<> dist {0, 10000};
        auto rng = std::bind(dist, random_engine);

        for (int i = 0; i < length; i++) {
//        result.AddNumber(dist(random_engine));
            result.AddNumber(rng());
        }

        return result;
    }

    NumberSeries NumberSeries::operator+(const NumberSeries &other) {
        NumberSeries result{};

        int a = series.size();
        int b = other.series.size();

        int min_length = a < b ? a : b;

        int i = 0;
        for (; i < min_length; i++) {
            result.AddNumber(series[i] + other.series[i]);
        }

        for (; i < series.size(); i++) {
            result.AddNumber(series[i]);
        }
```

```

        for (; i < other.series.size(); i++) {
            result.AddNumber(other.series[i]);
        }

        return result;
    }

    NumberSeries &NumberSeries::operator+=(const NumberSeries &other) {
        int i = 0;
        for (; i < series.size() && i < other.series.size(); i++) {
            series[i] += other.series[i];
            if (series[i] > maximum) {
                maximum = series[i];
            }
            else if (series[i] < minimum) {
                minimum = series[i];
            }
        }

        for (; i < other.series.size(); i++) {
            AddNumber(other.series[i]);
        }

        return *this;
    }

    bool NumberSeries::operator<(const NumberSeries &other) const {
        int my_amplitude = maximum - minimum;
        int other_amplitude = other.maximum - other.minimum;

        return my_amplitude < other_amplitude;
    }

    std::ostream& operator<<(std::ostream& os, const NumberSeries& numberSeries) {
        if (numberSeries.series.empty()) {
            os << "(empty series)";
        }
        else {
            for (const auto& i : numberSeries.series) {
                os << i << " ";
            }
        }

        return os;
    }
}
```

Listing 3: NumberSeriesWrapper.cpp

```
//
// Created by ane on 18/02/2022.
//

#include "NumberSeriesWrapper.h"

namespace series {
    int NumberSeriesWrapper::GetMaximum() {
        return ptr->GetMaximum();
    }

    int NumberSeriesWrapper::GetMinimum() {
```

```
        return ptr->GetMimimum();
    }

    NumberSeriesWrapper NumberSeriesWrapper::MakeRandom(int length) {
        return NumberSeriesWrapper(NumberSeries::MakeRandom(length));
    }

    NumberSeriesWrapper NumberSeriesWrapper::operator+(const NumberSeriesWrapper& other) {
        return NumberSeriesWrapper(*ptr + *other.ptr);
    }

    NumberSeriesWrapper &NumberSeriesWrapper::operator+=(const NumberSeriesWrapper& other) {
        *ptr += *other.ptr;
        return *this;
    }
}
```

Listing 4: NumberSeriesWrapper.h

```
//
// Created by ane on 18/02/2022.
//

#ifndef SESSION_5_NUMBERSERIESWRAPPER_H
#define SESSION_5_NUMBERSERIESWRAPPER_H


#include <memory>
#include "NumberSeries.h"

namespace series {
    class NumberSeriesWrapper {
        std::unique_ptr<NumberSeries> ptr{};
    public:

        explicit NumberSeriesWrapper(const NumberSeries& series) {
            ptr = std::make_unique<NumberSeries>(series);
        }

        NumberSeriesWrapper() = default;

        ~NumberSeriesWrapper() noexcept = default;

        NumberSeriesWrapper(const NumberSeriesWrapper& other) = delete;

        NumberSeriesWrapper& operator=(const NumberSeriesWrapper& other) = delete;

        NumberSeriesWrapper(NumberSeriesWrapper&& other) noexcept {
            *this = std::move(other);
        }

        NumberSeriesWrapper& operator=(NumberSeriesWrapper&& other) noexcept {
            if (this == &other) {
                return *this;
            }

//        std::swap(ptr, other.ptr);
            ptr = std::move(other.ptr);

            return *this;
        }

```

```
        int GetMaximum();

        int GetMinimum();

        static NumberSeriesWrapper MakeRandom(int);

        NumberSeriesWrapper operator+(const NumberSeriesWrapper&);

        NumberSeriesWrapper& operator+=(const NumberSeriesWrapper&);

        bool operator<(const NumberSeriesWrapper& other) const {
            return *ptr < *other.ptr;
        }
    };
}



#endif //SESSION_5_NUMBERSERIESWRAPPER_H
```

Listing 5: main.cpp

```
#include <iostream>
#include <vector>
#include <chrono>
#include "NumberSeries.h"
#include "NumberSeriesWrapper.h"
#include <numeric>

#define NUM_SERIES 100000
#define NUM_ELEMENTS 100

using namespace std;
using namespace series;

vector<NumberSeries> numberSeries(NUM_SERIES);
vector<NumberSeriesWrapper> numberSeriesWrappers(NUM_SERIES);

void testProgram() {

    cout << "Series: " << endl;

    auto first = std::chrono::high_resolution_clock::now();
    for (auto& series : numberSeries) {
        series = NumberSeries::MakeRandom(NUM_ELEMENTS);
    }
    auto last = std::chrono::high_resolution_clock::now();

    cout << "Building: " << chrono::duration<double, milli>(last - first).count() << endl;

    first = std::chrono::high_resolution_clock::now();
    std::sort(numberSeries.begin(), numberSeries.end());
    last = std::chrono::high_resolution_clock::now();

    cout << "Time: " << chrono::duration<double, milli>(last - first).count() << endl;
//    auto sum = std::accumulate(numberSeries.begin(), numberSeries.end(), NumberSeries());
//    cout << sum.GetMaximum() << endl;

    cout << endl;

    cout << "Wrappers: " << endl;
```

```
    first = std::chrono::high_resolution_clock::now();
    for (auto& series : numberSeriesWrappers) {
        series = NumberSeriesWrapper{NumberSeriesWrapper::MakeRandom(NUM_ELEMENTS)};
    }
    last = std::chrono::high_resolution_clock::now();

    cout << "Building: " << chrono::duration<double, milli>(last - first).count() << endl;

    first = std::chrono::high_resolution_clock::now();
    std::sort(numberSeriesWrappers.begin(), numberSeriesWrappers.end());
    last = std::chrono::high_resolution_clock::now();


    cout << "Time: " << chrono::duration<double, milli>(last - first).count() << endl;
}

int main() {
    testProgram();

    return 0;
}
```

Listing 6: CMakeLists.txt

```
cmake_minimum_required(VERSION 3.21)
project(Session_5)

set(CMAKE_CXX_STANDARD 20)

add_executable(Session_5 main.cpp NumberSeries.cpp NumberSeries.h NumberSeriesWrapper.cpp
↪NumberSeriesWrapper.h)
```