# Number Series Assignment

Christian Bager Bach Houmann

February 17, 2023

Listing 1: CMakeLists.txt

```
cmake_minimum_required(VERSION 3.24)
project(NumberSeries)

set(CMAKE_CXX_STANDARD 20)

add_executable(NumberSeries main.cpp number_series.cpp number_series.h series_wrapper.cpp series_wrapper.h)
```

Listing 2: series_wrapper.h

```
#ifndef SERIESWRAPPER_H
#define SERIESWRAPPER_H

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

namespace series
{
    class series_wrapper
    {
    public:
        series_wrapper() : _series(std::make_unique<number_series>()){};
        series_wrapper(const series_wrapper &other) : _series(std::make_unique<number_series>(*other._series)){};
        series_wrapper(const series_wrapper &&other) : _series(std::make_unique<number_series>(std::move(*other._series))){};

        series_wrapper &operator=(const series_wrapper &other);
        series_wrapper &operator=(series_wrapper &&other);

        static series_wrapper make_random(int lower, int upper, size_t length);

        int get_min() const;
        int get_max() const;

        series_wrapper operator+(const series_wrapper &other);
        series_wrapper &operator+=(const series_wrapper &other);

        bool operator<(const series_wrapper &other) const;

    private:
        std::unique_ptr<number_series> _series;
    };
}

#endif
```

Listing 3: number_series.h

```cpp
#ifndef NUMBERSERIES_H
#define NUMBERSERIES_H

#include <vector>
#include <iostream>
#include <algorithm>

namespace series
{
    class number_series
    {
    public:
        int get_min() const;

        int get_max() const;

        static number_series make_random(int upper, int lower, size_t length);

        number_series operator+(const number_series &other);

        number_series &operator+=(const number_series &other);

        bool operator<(const number_series &other) const;

    private:
        int averages[12]; // number_series sort just got slower
        std::vector<int> series;

        void update_min_max();

        int _minimum{};
        int _maximum{};
    };
}

#endif
```

Listing 4: series_wrapper.cpp

```cpp
#include "series_wrapper.h"

namespace series
{
    series_wrapper &series_wrapper::operator=(const series_wrapper &other)
    {
        if (this != &other)
            _series = std::make_unique<number_series>(*other._series);

        return *this;
    }

    series_wrapper &series_wrapper::operator=(series_wrapper &&other)
    {
        if (this != &other)
            _series = std::move(other._series);

        return *this;
    }

    series_wrapper series_wrapper::make_random(int lower, int upper, size_t length)
    {
        series_wrapper sw;
```

```
        sw._series->make_random(upper, lower, length);

        return sw;
    }

    int series_wrapper::get_min() const
    {
        return _series->get_min();
    }

    int series_wrapper::get_max() const
    {
        return _series->get_max();
    }

    series_wrapper series_wrapper::operator+(const series_wrapper &other)
    {
        series_wrapper sw{};
        sw._series = std::make_unique<number_series>(*_series + *other._series);

        return sw;
    }

    series_wrapper &series_wrapper::operator+=(const series_wrapper &other)
    {
        *_series += *other._series;

        return *this;
    }

    bool series_wrapper::operator<(const series_wrapper &other) const
    {
        return *_series < *other._series;
    }
}
```

Listing 5: number_series.cpp

```
#include "number_series.h"
#include <iterator>
#include <iostream>
#include <random>
#include <algorithm>
#include <functional>
#include <immintrin.h>

namespace series
{
    number_series number_series::make_random(int lower, int upper, size_t length)
    {
        number_series ns{};
        ns.series.reserve(length);

        std::random_device rd;
        std::mt19937 gen(rd());
        std::uniform_int_distribution<int> dist(lower, upper);

        std::generate(ns.series.begin(), ns.series.end(), [&gen, &dist]()
                      { return dist(gen); });

        ns.update_min_max();

```

```
25          return ns;
26      }
27
28      int number_series::get_min() const
29      {
30          return _minimum;
31      }
32
33      int number_series::get_max() const
34      {
35          return _maximum;
36      }
37
38      number_series number_series::operator+(const number_series &other)
39      {
40          if (series.size() != other.series.size())
41              throw std::invalid_argument("Vectors must have the same size");
42
43          number_series ns{};
44          ns.series.reserve(series.size());
45
46          for (size_t i = 0; i < series.size(); ++i)
47          {
48              ns.series.push_back(series[i] + other.series[i]);
49          }
50
51          ns._minimum = _minimum + other._minimum;
52          ns._maximum = _maximum + other._maximum;
53
54          return ns;
55      }
56
57      number_series &number_series::operator+=(const number_series &other)
58      {
59          if (series.size() != other.series.size())
60              throw std::invalid_argument("Vectors must have the same size");
61
62          for (size_t i = 0; i < series.size(); ++i)
63              series[i] += other.series[i];
64
65          update_min_max();
66
67          return *this;
68      }
69
70      bool number_series::operator<(const number_series &other) const
71      {
72          return (get_max() - get_min()) < (other.get_max() - other.get_min());
73      }
74
75      void number_series::update_min_max()
76      {
77          _minimum = 0;
78          _maximum = 0;
79
80          for (const auto &x : series)
81          {
82              if (x < _minimum)
83                  _minimum = x;
84
85              if (x < _maximum)
```

```
                _maximum = x;
        }
    }
}
```

Listing 6: main.cpp

```
#include "number_series.h"
#include "series_wrapper.h"

#include <iostream>
#include <vector>
#include <algorithm>
#include <chrono>

using namespace series;

void run();
void run_wrapper();

constexpr int size = 100000;

int main()
{
    run();

    run_wrapper();

    return 0;
}

void run()
{
    std::vector<number_series> nss(size);

    for (size_t i = 0; i < size; ++i)
        nss[i] = number_series::make_random(1, 10, 100);

    for (size_t i = 0; i < size; ++i)
        nss[i] += number_series::make_random(1, 10, 100);

    auto t_start = std::chrono::high_resolution_clock::now();
    std::sort(nss.begin(), nss.end(), [](const number_series &x, const number_series &y)
              { return x < y; });

    auto t_stop = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration<double, std::milli>(t_stop - t_start).count();

    std::cout << "Time to sort: \t\t" << duration << "ms." << std::endl;
}

void run_wrapper()
{
    std::vector<series_wrapper> nss(size);

    for (size_t i = 0; i < size; ++i) {
        nss.push_back(series_wrapper::make_random(1, 10, 100));
    }

    for (size_t i = 0; i < size; ++i)
    {
        auto random = series_wrapper::make_random(1, 10, 100);
```

```
56          nss.at(i) += random;
57      }
58      auto t_start = std::chrono::high_resolution_clock::now();
59      std::sort(nss.begin(), nss.end(), [](const series_wrapper &x, const series_wrapper &y)
60                { return x < y; });
61
62      auto t_stop = std::chrono::high_resolution_clock::now();
63      auto duration = std::chrono::duration<double, std::milli>(t_stop - t_start).count();
64
65      std::cout << "Time to sort wrapper: \t" << duration << "ms." << std::endl;
66  }
```