# number series

Jens Emil Fink H øjriis

February 24, 2024

Listing 1: ./number_series.hpp

```cpp
/// Created by Marius Mikucionis <marius@cs.aau.dk>
/**
 * Definitions of number_series and number_series_wrap classes.
 */

#ifndef INCLUDE_NUMBER_SERIES_H
#define INCLUDE_NUMBER_SERIES_H

#include <vector>
#include <limits>
#include <memory>
#include <iostream>

namespace data_series
{
    class number_series
    {
        std::vector<int> _data;
        int _min;
        int _max;

    public:
        number_series();
        number_series(std::initializer_list<int> s);

        size_t size() const noexcept;

        int get_min() const noexcept;

        int get_max() const noexcept;

        void add_value(int value);

        static number_series make_random(size_t length);

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

        int amplitude() const noexcept;

        friend number_series operator+(const number_series& lhs, const number_series& rhs);

        friend bool operator<(const number_series& lhs, const number_series& rhs);
    };

    class number_series_wrap
    {
        std::unique_ptr<number_series> _ns;
        number_series_wrap(number_series ns);
    public:
```

```
50          number_series_wrap();
51          number_series_wrap(std::initializer_list<int> s);
52          number_series_wrap(const number_series_wrap& other);
53          number_series_wrap(number_series_wrap&& other) noexcept;
54          number_series_wrap& operator=(const number_series_wrap& other);
55          number_series_wrap& operator=(number_series_wrap&& other) noexcept;
56
57          size_t size() const noexcept;
58
59          int get_min() const noexcept;
60
61          int get_max() const noexcept;
62
63          void add_value(int value);
64
65          int amplitude() const noexcept;
66
67          number_series_wrap& operator+=(const number_series_wrap& other);
68
69          friend number_series_wrap operator+(const number_series_wrap& lhs, const
    ↪number_series_wrap& rhs);
70
71          friend bool operator<(const number_series_wrap& lhs, const number_series_wrap& rhs);
72
73          static number_series_wrap make_random(size_t length);
74
75          // TODO: add the same interface as in number_series which forwards all the calls
76      };
77
78  }  // namespace data_series
79
80  #endif  // INCLUDE_NUMBER_SERIES_H
```

Listing 2: ./main.cpp

```
1   /// Created by Marius Mikucionis <marius@cs.aau.dk>
2   /**
3    * Purpose: compare the performance of number_series and number_series_wrap.
4    */
5   #include "number_series.hpp"
6
7   #include <chrono>
8
9   constexpr auto ns_number = 100'000;
10  constexpr auto ns_length = 100;
11
12  int main()
13  {
14      using namespace data_series;
15      using clk = std::chrono::high_resolution_clock;
16      using std::chrono::duration;
17      using std::milli;
18      using std::cout;
19      using std::endl;
20
21      // Part 2
22      std::vector<number_series> vv;
23      vv.reserve(ns_number);  // preallocate memory
24
25      // TODO: Populate the vv with data here
26
27      auto t0 = clk::now();
```

```
    // TODO: std::sort(vv.begin(), vv.end());
    auto t1 = clk::now();
    cout << "Sorting values: " << duration<double, milli>(t1 - t0).count() << " ms\n";

    // Part 3
    // Note that this is the exact same code as for Part 1 except using number_series_wrap
    auto vw = std::vector<number_series_wrap>{};
    vw.reserve(ns_number);  // preallocate memory

    // TODO: Populate the vw with data here

    t0 = clk::now();
    // TODO: std::sort(vw.begin(), vw.end());
    t1 = clk::now();
    cout << "Sorting wrapped pointers: " << duration<double, milli>(t1 - t0).count() << " ms\n";

    cout << "sizeof(number_series): " << sizeof(number_series) << '\n';
}

/**
Important: measure the optimized ("Release") build!

Sample result:

Sorting values: XXX ms
Sorting wrapped pointers: YYY ms
sizeof(number_series): ZZZ

Interpretation:

Part 3 is about XX% ****er than Part 2.
PUT YOUR CONCLUSION HERE

Sample result, if number_series is padded with array:

Sorting values: XXX ms
Sorting wrapped pointers: YYY ms
sizeof(number_series): ZZZ

Part 3 is about XX% ****er than Part 2.
PUT YOUR CONCLUSION HERE

*/
```

Listing 3: ./number_series.cpp

```
/// Created by Marius Mikucionis <marius@cs.aau.dk>
/**
 * Definitions/implementation of some number_series methods
 */

#include "number_series.hpp"
#include <random>
#include <memory>

namespace data_series
{
    number_series_wrap::number_series_wrap() {
        _ns = std::make_unique<number_series>();
    }

    number_series_wrap::number_series_wrap(std::initializer_list<int> s) {
```

```
        _ns = std::make_unique<number_series>(std::move(s));
    }

    number_series_wrap::number_series_wrap(const number_series_wrap& other) {
        _ns = std::make_unique<number_series>(*other._ns);
    }

    number_series_wrap::number_series_wrap(number_series_wrap&& other) noexcept {
        _ns = std::move(other._ns);
        other._ns = std::make_unique<number_series>();
    }

    number_series_wrap& number_series_wrap::operator=(const number_series_wrap& other)
    {
        _ns = std::make_unique<number_series>(*other._ns);
        return *this;
    }

    number_series_wrap& number_series_wrap::operator=(number_series_wrap&& other) noexcept
    {
        _ns = std::move(other._ns);
        other._ns = std::make_unique<number_series>();
        return *this;
    }

    size_t number_series_wrap::size() const noexcept {
        return _ns->size();
    }

    int number_series_wrap::get_min() const noexcept {
        return _ns->get_min();
    }

    int number_series_wrap::get_max() const noexcept {
        return _ns->get_max();
    }

    void number_series_wrap::add_value(int value) {
        _ns->add_value(value);
    }

    int number_series_wrap::amplitude() const noexcept {
        return _ns->amplitude();
    }

    number_series_wrap& number_series_wrap::operator+=(const number_series_wrap& other)
    {
        *_ns += *other._ns;
        return *this;
    }

    number_series_wrap number_series_wrap::make_random(size_t length){
        return number_series_wrap(number_series::make_random(length));
    }

    number_series_wrap::number_series_wrap(number_series ns) {
        _ns = std::make_unique<number_series>(std::move(ns));
    }

    number_series::number_series() {
        _min = INT_MAX;
```

```
78          _max = INT_MIN;
79      }
80
81      number_series::number_series(std::initializer_list<int> s) {
82          _min = INT_MAX;
83          _max = INT_MIN;
84          _data.reserve(s.size());
85          for (auto n : s) {
86              add_value(n);
87          }
88      }
89
90      size_t number_series::size() const noexcept {
91          return _data.size();
92      }
93
94      int number_series::get_min() const noexcept {
95          return _min;
96      }
97
98      int number_series::get_max() const noexcept {
99          return _max;
100     }
101
102     void number_series::add_value(int value)
103     {
104         _data.push_back(value);
105         _min = std::min(value, _min);
106         _max = std::max(value, _max);
107     }
108
109     number_series number_series::make_random(size_t length) {
110         std::random_device rd;
111         std::mt19937 gen(rd());
112         std::uniform_int_distribution<> distrib(INT_MIN, INT_MAX);
113         number_series series;
114
115         series._data.reserve(length);
116
117         for (size_t i = 0; i < length; i++) {
118             series.add_value(distrib(gen));
119         }
120
121         return series;
122     }
123
124     number_series& number_series::operator+=(const number_series& other)
125     {
126         auto tit = _data.begin();
127         auto oit = other._data.begin();
128         for (; tit != _data.end() && oit != other._data.end(); tit++, oit++) {
129             *tit += *oit;
130         }
131         for (; oit != other._data.end(); oit++) {
132             add_value(*oit);
133         }
134         return *this;
135     }
136
137     int number_series::amplitude() const noexcept {
138         return (_max - _min);
```

```
139     }
140
141     number_series operator+(const number_series& lhs, const number_series& rhs) {
142         number_series rv = lhs;
143         rv += rhs;
144         return rv;
145     }
146
147     bool operator<(const number_series& lhs, const number_series& rhs) {
148         return lhs.amplitude() < rhs.amplitude();
149     }
150
151     number_series_wrap operator+(const number_series_wrap& lhs, const number_series_wrap& rhs)
152     {
153         return number_series_wrap(*lhs._ns + *rhs._ns);
154     }
155
156     bool operator<(const number_series_wrap& lhs, const number_series_wrap& rhs) {
157         return *lhs._ns < *rhs._ns;
158     }
159
160 }  // namespace data_series
```

Listing 4: ./number_series_bm.cpp

```
1  /// Created by Marius Mikucionis <marius@cs.aau.dk>
2  #include "number_series.hpp"
3  #include <benchmark/benchmark.h>
4
5  auto make_vv(size_t ns_number, size_t ns_length)
6  {
7      using data_series::number_series;
8      auto vv = std::vector<number_series>{};
9      vv.reserve(ns_number);  // preallocate memory
10
11     for (size_t i = 0; i < ns_number; i++) {
12         vv.push_back(number_series::make_random(ns_length));
13     }
14
15     return vv;
16 };
17
18 auto make_wrapper(size_t ns_number, size_t ns_length)
19 {
20     using data_series::number_series_wrap;
21     auto vw = std::vector<number_series_wrap>{};
22     vw.reserve(ns_number);  // preallocate memory
23
24     for (size_t i = 0; i < ns_number; i++) {
25         vw.push_back(number_series_wrap::make_random(ns_length));
26     }
27
28     return vw;
29 };
30
31 static void bm_ns_sort(benchmark::State& state)
32 {
33     const auto ns_number = state.range(0);
34     const auto ns_length = state.range(1);
35     const auto input = make_vv(ns_number, ns_length);
36     for (auto _ : state) {
37         state.PauseTiming();
```

```
        auto vv = input;
        state.ResumeTiming();
        std::sort(vv.begin(), vv.end());
        benchmark::DoNotOptimize(vv.data());  // tells compiler that vv.data() is useful
        benchmark::ClobberMemory();           // flush changes to memory
    }
}
BENCHMARK(bm_ns_sort)->ArgPair(100'000, 100);

static void bm_ns_wrap_sort(benchmark::State& state)
{
    const auto ns_number = state.range(0);
    const auto ns_length = state.range(1);
    const auto input = make_wrapper(ns_number, ns_length);
    for (auto _ : state) {
        state.PauseTiming();
        auto vw = input;
        state.ResumeTiming();
        std::sort(vw.begin(), vw.end());
        benchmark::DoNotOptimize(vw.data());  // tells compiler that vw.data() is useful
        benchmark::ClobberMemory();           // flush changes to memory
    }
}
BENCHMARK(bm_ns_wrap_sort)->ArgPair(100'000, 100);
```

Listing 5: ./number_series_test.cpp

```
/// Created by Marius Mikucionis <marius@cs.aau.dk>

/** Unit tests for number_series */

#include "number_series.hpp"

#include <doctest/doctest.h>

using namespace data_series;

/// number_series class
TEST_CASE("Maintain minimum and maximum values")
{
    auto ns = number_series{};
    // TODO: uncomment one test at a time, implement it and check it
    ns.add_value(10);
    CHECK(ns.get_min() == 10);
    CHECK(ns.get_max() == 10);
    SUBCASE("Add greater")
    {
        ns.add_value(15);
        CHECK(ns.get_min() == 10);
        CHECK(ns.get_max() == 15);
        ns.add_value(17);
        CHECK(ns.get_min() == 10);
        CHECK(ns.get_max() == 17);
        ns.add_value(13);
        CHECK(ns.get_min() == 10);
        CHECK(ns.get_max() == 17);
    }
    SUBCASE("Add lesser")
    {
        ns.add_value(5);
        CHECK(ns.get_min() == 5);
        CHECK(ns.get_max() == 10);
```

```
        ns.add_value(3);
        CHECK(ns.get_min() == 3);
        CHECK(ns.get_max() == 10);
        ns.add_value(7);
        CHECK(ns.get_min() == 3);
        CHECK(ns.get_max() == 10);
    }

}

// TODO: uncomment one test at a time, then implement it
TEST_CASE("Special members: ctors, dtor, assignment")
{
    const auto ns1 = number_series{11, 3, 7};
    CHECK(ns1.size() == 3);
    CHECK(ns1.get_min() == 3);
    CHECK(ns1.get_max() == 11);
    auto ns2 = number_series{27, 20, 33, 23};
    CHECK(ns2.size() == 4);
    CHECK(ns2.get_min() == 20);
    CHECK(ns2.get_max() == 33);
    auto ns3 = ns1;
    CHECK(ns3.size() == 3);
    CHECK(ns3.get_min() == 3);
    CHECK(ns3.get_max() == 11);
    ns2 = std::move(ns3);
    CHECK(ns2.size() == 3);
    CHECK(ns2.get_min() == 3);
    CHECK(ns2.get_max() == 11);
    CHECK(ns3.size() == 0); // your implementation may differ
}

// TODO: uncomment one test at a time, then implement it
TEST_CASE("Static factory method")
{
    auto ns = number_series::make_random(4);
    CHECK(ns.size() == 4);
}


// TODO: uncomment one test at a time, then implement it
TEST_CASE("operator+ and operator+= over number series")
{
    auto ns1 = number_series::make_random(2);
    CHECK(ns1.size() == 2);
    auto ns2 = number_series::make_random(3);
    CHECK(ns2.size() == 3);
    auto ns3 = ns1 + ns2;
    CHECK(ns1.size() == 2);
    CHECK(ns2.size() == 3);
    CHECK(ns3.size() == 3);
    ns2.add_value(10);
    CHECK(ns2.size() == 4);
    (ns3 += ns1) += ns2;
    CHECK(ns1.size() == 2);
    CHECK(ns2.size() == 4);
    CHECK(ns3.size() == 4);
}


// TODO: uncomment one test at a time, then implement it
```

```
97  TEST_CASE("operator< using amplitudes")
98  {
99      auto ns1 = number_series{6, 3, 9};
100     CHECK(ns1.amplitude() == 6);
101     auto ns2 = number_series{24, 21, 22};
102     CHECK(ns2.amplitude() == 3);
103     CHECK(ns2 < ns1);
104 }
105
106
107 /// number_series_wrap class
108 // TODO: uncomment one test at a time, then implement it
109 TEST_CASE("Maintain minimum and maximum values")
110 {
111     auto ns = number_series_wrap{};
112     ns.add_value(10);
113     CHECK(ns.get_min() == 10);
114     CHECK(ns.get_max() == 10);
115     SUBCASE("Add greater")
116     {
117         ns.add_value(15);
118         CHECK(ns.get_min() == 10);
119         CHECK(ns.get_max() == 15);
120         ns.add_value(17);
121         CHECK(ns.get_min() == 10);
122         CHECK(ns.get_max() == 17);
123         ns.add_value(13);
124         CHECK(ns.get_min() == 10);
125         CHECK(ns.get_max() == 17);
126     }
127     SUBCASE("Add lesser")
128     {
129         ns.add_value(5);
130         CHECK(ns.get_min() == 5);
131         CHECK(ns.get_max() == 10);
132         ns.add_value(3);
133         CHECK(ns.get_min() == 3);
134         CHECK(ns.get_max() == 10);
135         ns.add_value(7);
136         CHECK(ns.get_min() == 3);
137         CHECK(ns.get_max() == 10);
138     }
139 }
140
141 // TODO: uncomment one test at a time, then implement it
142 TEST_CASE("Special members: ctors, dtor, assignment")
143 {
144     const auto ns1 = number_series_wrap{11, 3, 7};
145     CHECK(ns1.size() == 3);
146     CHECK(ns1.get_min() == 3);
147     CHECK(ns1.get_max() == 11);
148     auto ns2 = number_series_wrap{27, 20, 33, 23};
149     CHECK(ns2.size() == 4);
150     CHECK(ns2.get_min() == 20);
151     CHECK(ns2.get_max() == 33);
152     auto ns3 = ns1;
153     CHECK(ns3.size() == 3);
154     CHECK(ns3.get_min() == 3);
155     CHECK(ns3.get_max() == 11);
156     ns2 = std::move(ns3);
157     CHECK(ns2.size() == 3);
```

```
158     CHECK(ns2.get_min() == 3);
159     CHECK(ns2.get_max() == 11);
160     CHECK(ns3.size() == 0); // your implementation may differ
161 }
162
163 // TODO: uncomment one test at a time, then implement it
164 TEST_CASE("Class should have a static factory method")
165 {
166     auto ns = number_series::make_random(4);
167     CHECK(ns.size() == 4);
168 }
169
170 // TODO: uncomment one test at a time, then implement it
171 TEST_CASE("operator+ and operator+= over number series")
172 {
173     auto ns1 = number_series_wrap::make_random(2);
174     CHECK(ns1.size() == 2);
175     auto ns2 = number_series_wrap::make_random(3);
176     CHECK(ns2.size() == 3);
177     auto ns3 = ns1 + ns2;
178     CHECK(ns1.size() == 2);
179     CHECK(ns2.size() == 3);
180     CHECK(ns3.size() == 3);
181     ns2.add_value(10);
182     CHECK(ns2.size() == 4);
183     (ns3 += ns1) += ns2;
184     CHECK(ns1.size() == 2);
185     CHECK(ns2.size() == 4);
186     CHECK(ns3.size() == 4);
187 }
188
189
190 // TODO: uncomment one test at a time, then implement it
191 TEST_CASE("operator< using amplitudes")
192 {
193     auto ns1 = number_series_wrap{6, 3, 9};
194     CHECK(ns1.amplitude() == 6);
195     auto ns2 = number_series_wrap{24, 21, 22};
196     CHECK(ns2.amplitude() == 3);
197     CHECK(ns2 < ns1);
198 }
```

Listing 6: ./sort_bm.cpp

```
1  /// Created by Marius Mikucionis <marius@cs.aau.dk>
2
3  #include <benchmark/benchmark.h>
4  #include <random>
5
6  auto make_data(size_t size, int max)
7  {
8      static auto gen = std::default_random_engine{std::random_device{}()};
9      static auto dist = std::uniform_int_distribution{0, max};
10     auto res = std::vector<int>(size);
11     std::generate(res.begin(), res.end(), [] { return dist(gen); });
12     return res;
13 };
14
15 static void bm_sort(benchmark::State& state)
16 {
17     const auto size = state.range(0);
18     const auto max = state.range(1);
```

```
    const auto data = make_data(size, max);
    for (auto _ : state) {
        state.PauseTiming();
        auto input = data;
        state.ResumeTiming();
        sort(input.begin(), input.end());
        benchmark::DoNotOptimize(input.data());  // tells compiler that vv.data() is useful
        benchmark::ClobberMemory();              // flush changes to memory
    }
}
BENCHMARK(bm_sort)->ArgPair(100, 100)->ArgPair(1'000, 100)->ArgPair(10'000,
↪100)->ArgPair(100'000, 100);
```

Listing 7: ./CMakeLists.txt

```
cmake_minimum_required(VERSION 3.15)
project(Extended1)

set(CMAKE_CXX_STANDARD 20)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
set(CMAKE_CXX_EXTENSIONS OFF)
set(CMAKE_EXPORT_COMPILE_COMMANDS ON)
set(CMAKE_POSITION_INDEPENDENT_CODE ON)
set(BUILD_SHARED_LIBS OFF)

include(sanitizers.cmake)
include(doctest.cmake)
include(benchmark.cmake)

enable_testing()

add_executable(extended1_main main.cpp number_series.cpp number_series.hpp)
add_test(NAME extended1_main COMMAND extended1_template_main)

add_executable(number_series_test number_series_test.cpp number_series.cpp number_series.hpp)
target_link_libraries(number_series_test PRIVATE doctest::doctest_with_main)
add_test(NAME number_series_test COMMAND number_series_test)

add_executable(sort_bm sort_bm.cpp)
target_link_libraries(sort_bm PRIVATE benchmark::benchmark_main)
add_test(NAME sort_bm COMMAND sort_bm)

add_executable(number_series_bm number_series_bm.cpp number_series.cpp number_series.hpp)
target_link_libraries(number_series_bm PRIVATE benchmark::benchmark_main)
add_test(NAME number_series_bm COMMAND number_series_bm)
```

Listing 8: ./README.md

```
# Number Series, Wrapper and Benchmark

In this assignment, you have to implement a data type (class) ‘number_series‘, to store a number
↪series (just a sequence of integers).

0. Check that the initial project works: select ‘All CTest‘ target and run, all tests should
↪pass. **Ask for help if something does not work**, see if you can work with
↪[main.cpp](main.cpp) instead, try running just ‘main‘.

1. Implement ‘number_series‘ class
    - Uncomment one test at a time in [number_series_test.cpp](number_series_test.cpp), recompile
↪and observe errors and test failures.
    - Implement missing functionality (solve ‘TODO:‘ comments) in
↪[number_series.hpp](number_series.hpp) until the test passes.
```

```
10      - Use ‘std::vector<int>‘ to model data.
11      - Maintain the **minimum** and the **maximum** values of a number series as members of your class (so that you do not have to compute them when needed).
12      - Implement the necessary *constructors/destructors/assignment operators*. Implement them only if the ones generated by default are not good. Study Section 17.6 or Item 17 in EMC++ to learn about what is generated by the compiler.
13      - Your class should have a **static** member factory function ‘make_random‘ that returns a random number series of a desired length. See [uniform_int_distribution example](https://en.cppreference.com/w/cpp/numeric/random/uniform_int_distribution) how to generate random numbers.
14      - Implement ‘operator+‘ and ‘operator+=‘ to add two number series element-wise. Decide yourself what to do if the number series have different lengths.
15      - Implement ‘operator<‘ to compare the *amplitudes* of two number series. The amplitude is the difference between the *maximum* and the *minimum* values.
16   2. Solve ‘TODO‘s in [number_series_bm.cpp](number_series_bm.cpp) benchmark and run ‘number_series_bm‘ target in ‘Release‘ profile:
17      - Fill a vector of ‘100’000‘ random number series, each with ‘100‘ elements.
18      - Add a random number series to each of number series in the vector.
19      - Sort the vector according to the amplitudes of number series.
20      - Record how much time it takes to sort (remember to write down the timings in comments).
21   3. Create a wrapper class, that has just one private data member: ‘std::unique_ptr<number_series>‘ and add the same interface as for ‘number_series‘. Think about the copy/move constructors/assignment operators, that you may need to provide or rely on the default ones. On the client side in [number_series_bm.cpp](number_series_bm.cpp), demonstrate that you can now use essentially the same code as in step 2.
22      - What do you expect regarding the performance of sorting objects of this wrapper class when compared to sorting the original ‘number_series‘ objects?
23      - Measure the performance and see whether your expectations were confirmed.
24   4. Add ‘int dummy[100];‘ as an extra data member of ‘number_series‘.
25      - Rerun ‘number_series_bm‘ again.
26      - How did the performance change? Why?
27
28   In your code, pay attention to **safety** and **performance**:
29   1. Use ‘const‘ whenever it makes sense (arguments, member functions).
30   2. Use pass-by-reference for efficiency whenever possible.
31   3. Avoid using plain pointers.
32   4. Use ‘Release‘ profile (without ‘Debug‘ info overhead) when benchmarking. To add ‘Release‘ profile in CLion visit ‘Settings‘ > ‘Build, Execution and Deployment‘ > ‘CMake‘ > ‘Profiles‘ and click ‘+‘, it should create ‘Release‘ profile automatically (no further tweaking should be needed).
33
34   ## Unit Tests
35   [number_series_test.cpp](number_series_test.cpp) includes unit tests to guide you through the implementation.
36
37   ## Benchmarks
38   [sort_bm.cpp](sort_bm.cpp) includes a sorting example demonstrating how to use Google Benchmark library.
39   [number_series_bm.cpp](number_series_bm.cpp) your code for benchmarking the ‘number_series‘ and ‘number_series_wrap‘.
40
41   ## Libraries included
42   - [doctest](https://github.com/doctest/doctest): unit test framework.
43   - [Google Benchmark](https://github.com/google/benchmark): microbenchmark support library.
44
45   CMake should fetch those libraries automatically during ‘cmake‘ reload, and the libraries should build automatically when building the project.
```