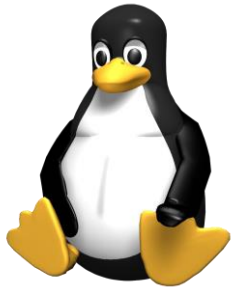
Algorithm intuition revisited

[about me]

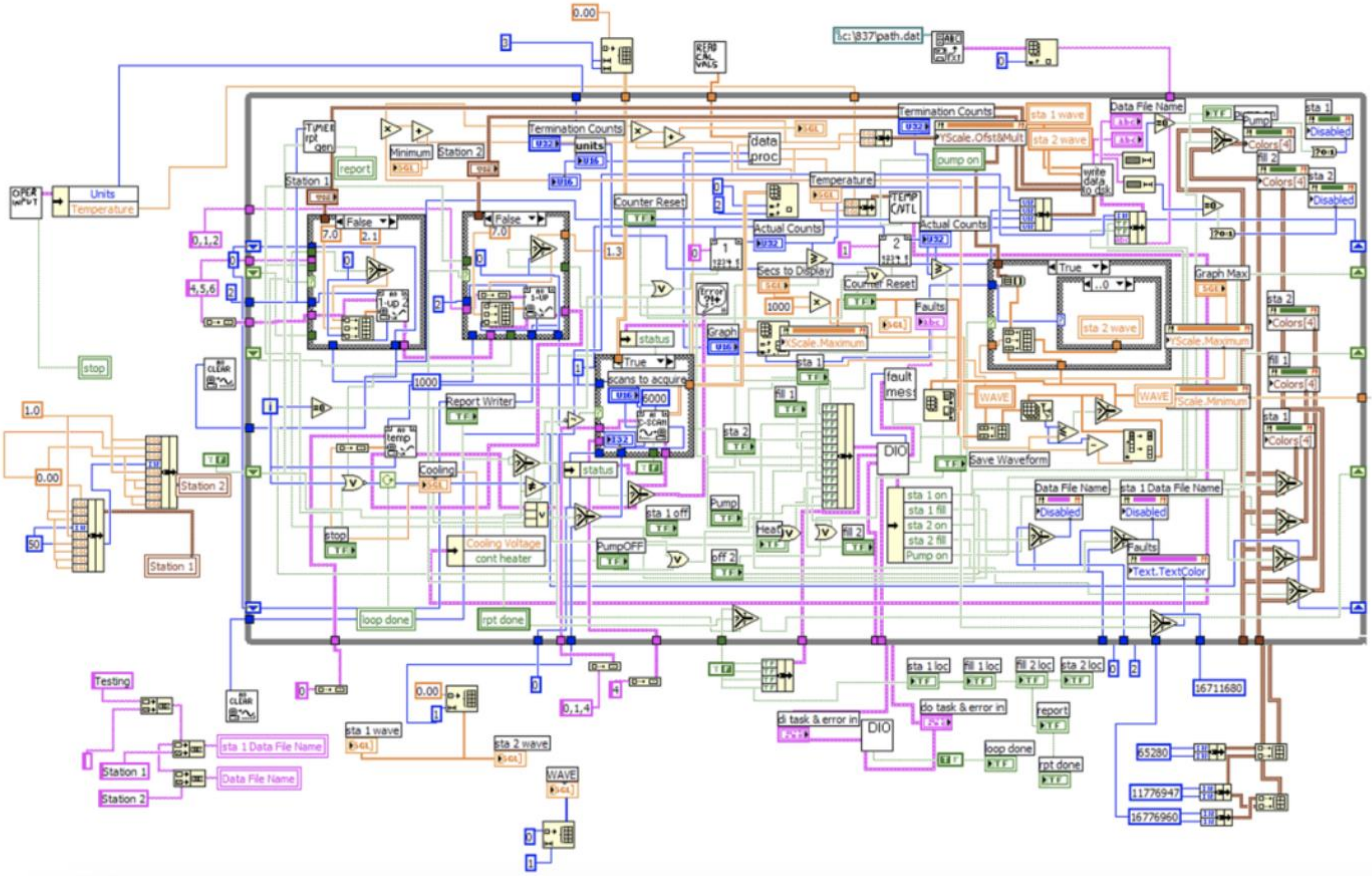
- Bruno Hendrickx
- Software Engineer at KLA
- Programming in C++ for ~10 years
- Father of 3
- Marathon runner

[about me]



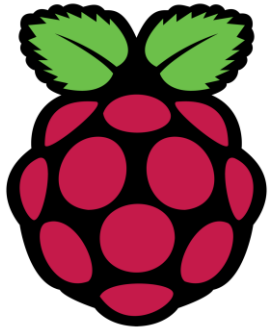
[about me]

[about me]

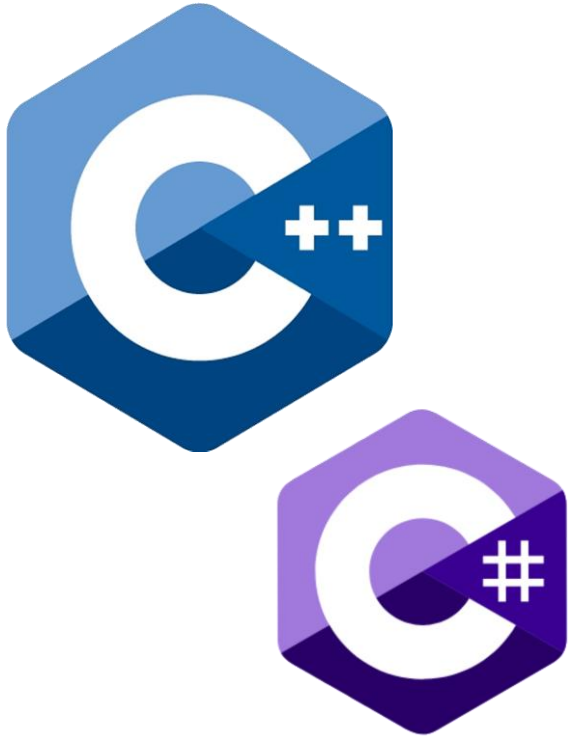


Termination Counts
Counter Reset
Actual Counts
Secs to Display
Temperature
data proc
Report Writer
scans to acquire
status
Cooling
Cooling Voltage
cont heater
Station 1
Station 2
Units
Temperature
stop
loop done
rpt done
Save Waveform
Data File Name
sta 1 Data File Name
Disabled
Faults
Text.TextColor
Graph Max
YScale.Maximum
XScale.Maximum
Scale.Minimum
Graph
fault mess
DIO
sta 1 on
sta 1 fill
sta 2 on
sta 2 fill
Pump on
PumpOFF
sta 1 off
Heat
report
Testing
sta 1 wave
sta 2 wave
WAVE
di task & error in
do task & error in
sta 1 loc
fill 1 loc
fill 2 loc
sta 2 loc
65280
11776947
16776960
16711680
c:\837\path.dat

[about me]



[about me]



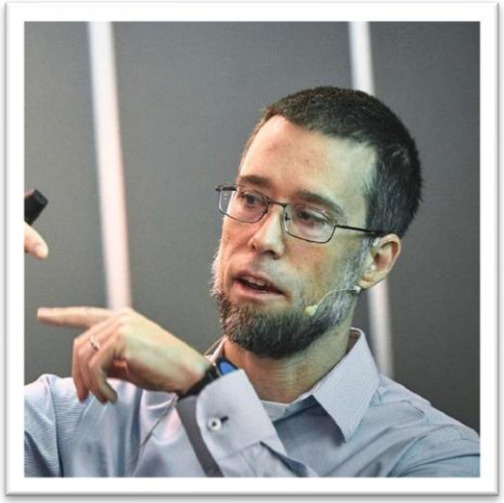
[highly recommended]

Cache-friendly data + functional + ranges = 

Trompeloeil, mocking framework for modern C++

Ceci n'est pas un objet.

[highly recommended]



Rich code for tiny machines: a simple commodore 64 game in C++17

C++ weekly youtube channel

Constexpr ALL the things

[disclaimer]

- Scott Schurr “I’m not an expert; I’m just a dude.”
- Slideware code
- I have opinions, feel free to share yours
- I’m speaking as myself

[background]

- Principal Scientist at Adobe Systems
- Github
- Numerous talks on concurrency, algorithms and software design
- C++ Seasoning talk

[background]

- Research scientist at Nivdia
- Host of ADSP
- Active conference speaker

[background]

- Father of the STL (Standard Template Library)
- Books:
  - Elements of programming
  - From mathematics to generic programming

[motivation]

- Develop algorithm intuition
- Sharing these insights
- Getting you excited about algorithms

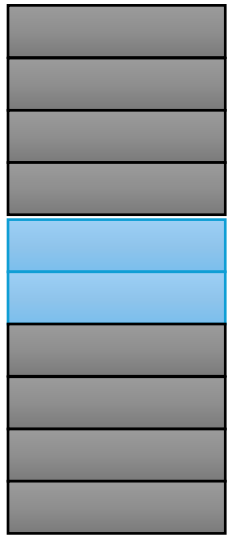
[motivation]

Quotes by Kate Gregory

“Simplicity is not for beginners.
Writing simple code, requires knowledge of the language.”

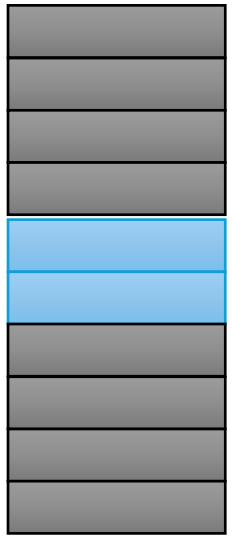
“… and just as you can say, that would be a good use of a linked list, we don’t have that intuition about algorithms yet.”

[question]



How do I move the blue elements upwards?

[question]



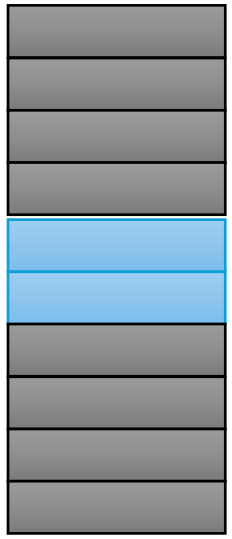
```
std::vector<int> seq {1,1,1,1,2,3,1,1};

auto first = seq[4];
seq.erase(std::begin(seq)+4);
seq.insert(std::begin(seq)+2, first);

auto second = seq[5];
seq.erase(std::begin(seq)+5);
seq.insert(std::begin(seq)+3, second);

//1,1,2,3,1,1,1,1
```

[question]



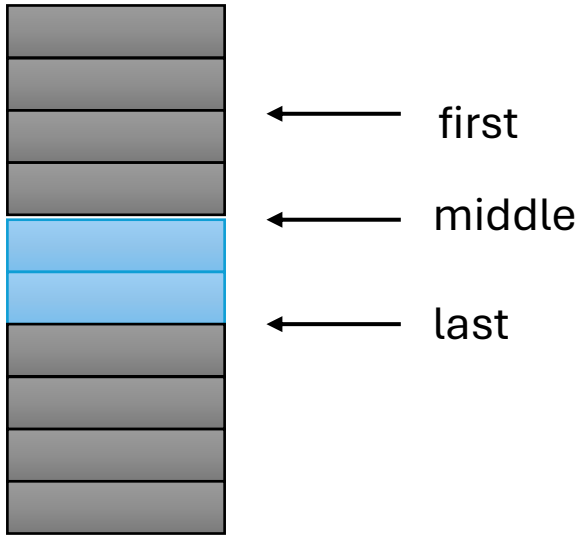
```
std::vector<int> seq {1,1,1,1,2,3,1,1};

std::rotate(std::begin(seq)+2, std::begin(seq)+4,
            std::begin(seq)+6);

//1,1,2,3,1,1,1,1
```

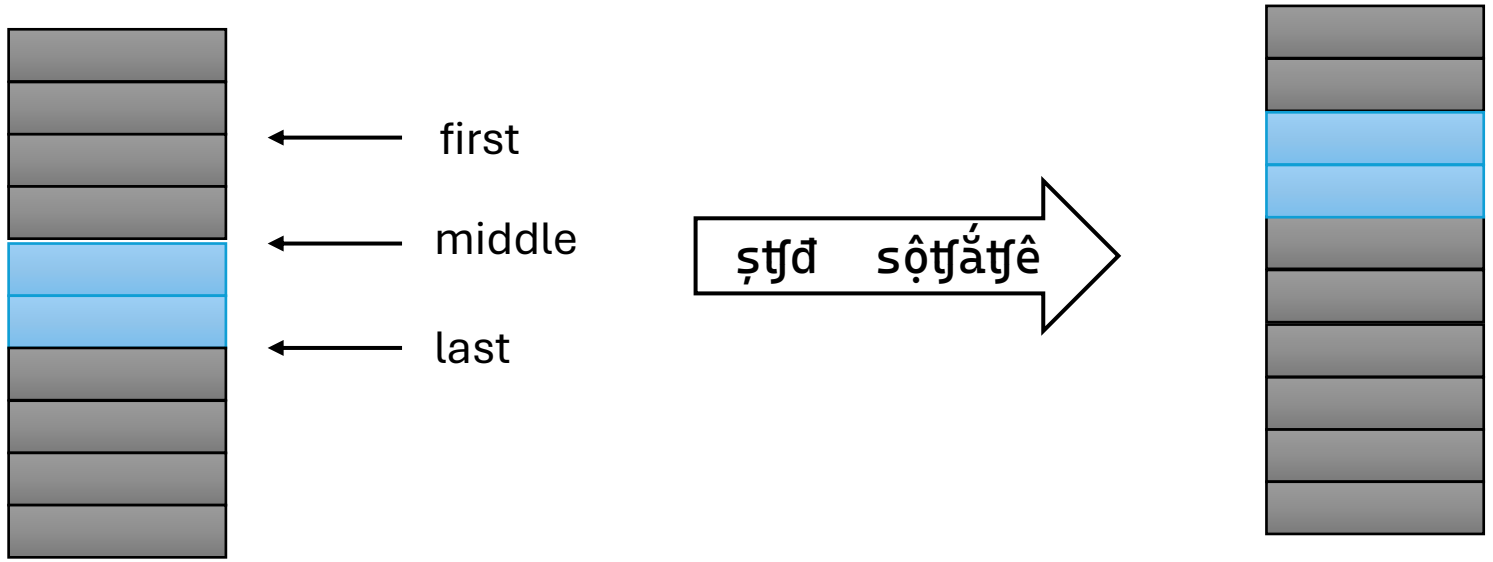
[std::rotate]

```
template<typename It>
It rotate(It first, It middle, It last);
```

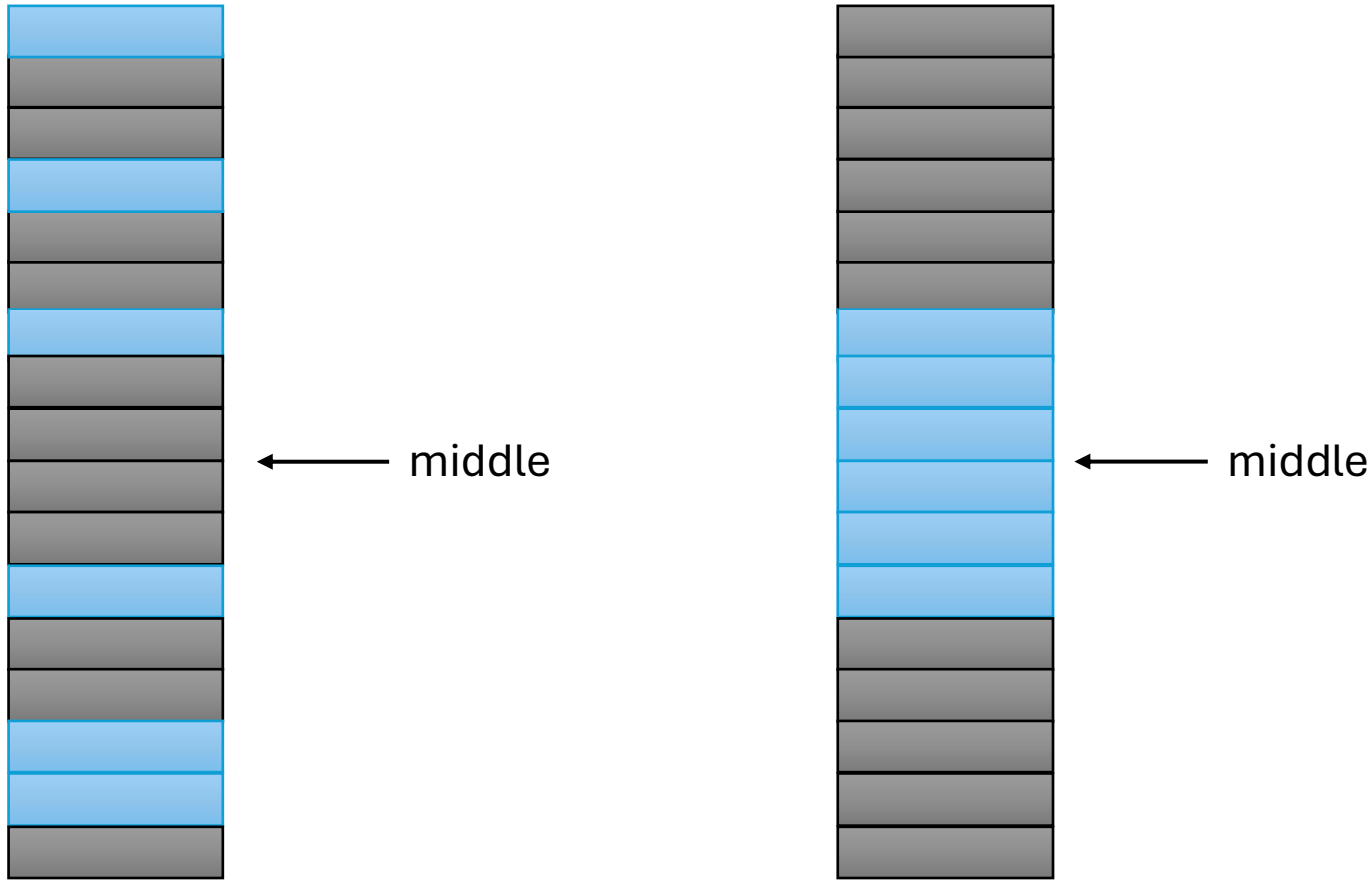


[std::rotate]

```
template<typename It>
It rotate(It first, It middle, It last);
```



[quiz]



[std::stable_partition]

```
template<typename It, typename U>
It stable_partition(It first, It last, U predicate);
```

```
std::vector<int> vec{1,2,3,4,5,6,7,8,9,10};
std::stable_partition(std::begin(vec), std::end(vec),
                      [](int value) -> bool {return value % 2 == 0});
```

[quiz]

HackerRank

You are given a long list of surface area’s. You want to know the accumulated surface area of the biggest 3.

[quiz]

```
auto first = [count = 3](std::vector<int>& surfaces) -> int {
    int sum = 0';
    for (int idx = 0; idx < count; ++idx) {
        auto max_it = std::max_element(std::begin(surfaces), std::end(surfaces));
        sum += *max_it;
        surfaces.erase(max_it);
    }
};
```

[quiz]

```
auto second = [](const std::vector<int>& surfaces) -> int {
    std::array<int, 3> cache {0,0,0};
    for (int surface : surfaces) {
        if (surface > cache[0]) {
            cache[2] = cache[1];
            cache[1] = cache[0];
            cache[0] = surface;
        }
    }
    return cache[0] + cache[1] + cache[2];
};
```

[quiz]

```
auto third = [count = 3](std::vector<int>& surfaces) -> int {
    std::sort(std::begin(surfaces), std::end(surfaces));
    return std::accumulate(std::end(surfaces)-count, std::end(surfaces), 0);
};
```

[quiz]

```
auto fourth = [count = 3](std::vector<int>& surfaces) -> int {
    std::nth_element(std::begin(surfaces), std::end(surfaces)-count, std::end(surfaces));
    return std::accumulate(std::end(surfaces)-count, std::end(surfaces), 0);
};
```

[std::nth_element]

```
template<typename It>
void nth_element(It first, It nth, It last);
```

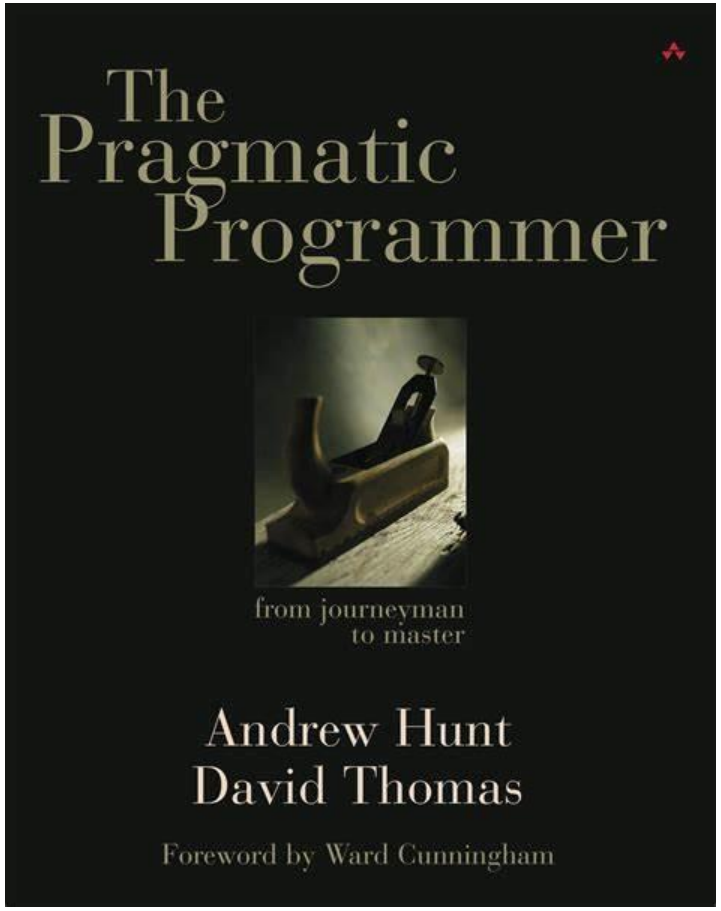
```
template<typename It, typename C>
It nth_element(It first, It nth, Ith last, C comparator);
```

[std::accumulate]

```
template<typename It, typename T>
T accumulate(It first, It last, T init);

template<typename It, typename T, typename B>
T accumulate(It first, It last, T init, B binaryOperation);
```

[digression]



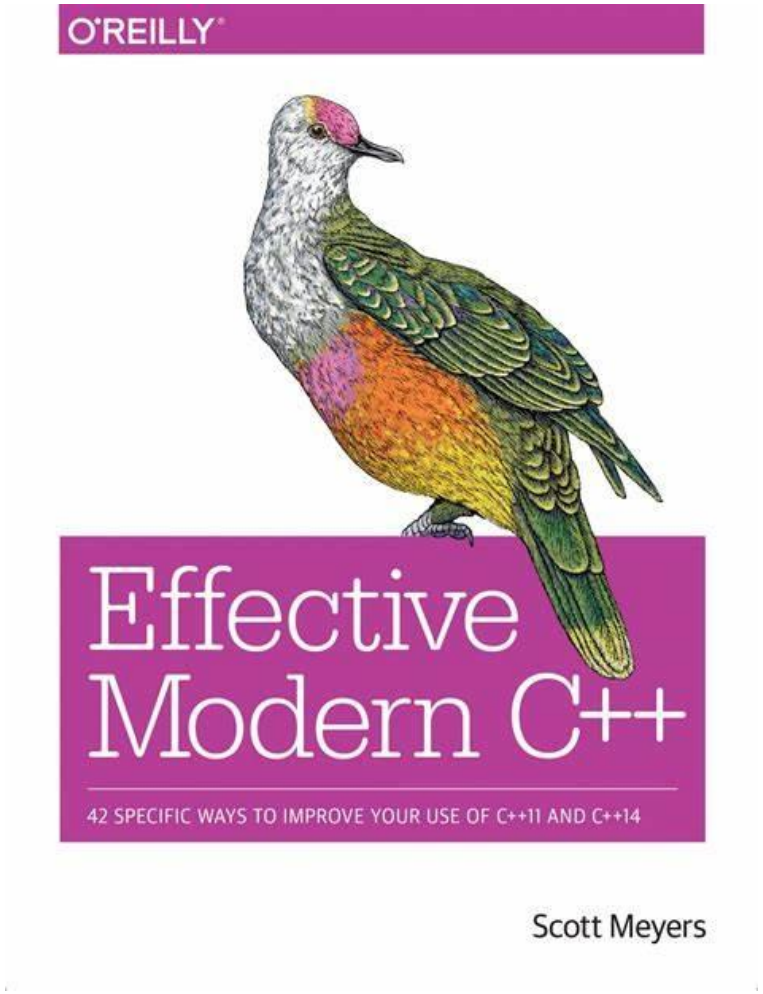
“Coding by coincidence”

[digression]

```
template<typename It, typename T, typename B>
T accumulate(It first, It last, T init, B binaryOperation);

template<typename It, typename T, typename B>
T reduce(It first, It last, T init, B binaryOperation);
```

[digression]


O'REILLY
Effective Modern C++
42 SPECIFIC WAYS TO IMPROVE YOUR USE OF C++11 AND C++14
Scott Meyers

[quiz]

Advent of code

You get a list of positive integers. If this list strictly increasing or decreasing, the result is positive.
Negative otherwise.

42, 44, 47, 49, 51, 52, 54, 52 => negative

80, 82, 85, 86, 87, 90, 94 => positive

[quiz]

```
auto first = [](const std::vector<int>& sequence) -> bool {
    std::vector<int> differences{};
    std::transform(std::cbegin(sequence), std::cend(sequence)-1, std::cbegin(sequence)+1,
        std::back_inserter(differences), [](int first, int second) -> int {
        return second - first;});

    int count = std::accumulate(std::cbegin(differences), std::cend(differences), 0,
        [](int acc, int diff) -> int {
        if (diff > 0) return ++acc;
        else if (diff < 0) return --acc;
        else return acc;
        });

    return std::abs(count) == differences.size();
};
```

[quiz]

```
auto second = [](const std::vector<int>& sequence) -> bool {
    std::vector<int> differences{};
    std::adjacent_difference(std::cbegin(sequence), std::cend(sequence),
        std::back_inserter(differences));

    auto my_less = std::bind(std::less{}, std::placeholders::_1, 0);
    auto my_greater = std::bind(std::greater{}, std::placeholders::_1, 0);

    return std::all_of(std::cbegin(differences)+1, std::cend(differences), my_less)
        || std::all_of(std::cbegin(differences)+1, std::cend(differences), my_greater);
};
```

[quiz]

```
auto third = [](const std::vector<int>& sequence) -> bool {
    if (auto it = std::adjacent_find(std::cbegin(sequence), std::cend(sequence));
        it == std::cend(sequence) {
        return std::is_sorted(std::cbegin(sequence), std::cend(sequence))
            || std::is_sorted(std::crbegin(sequence), std::crend(sequence));
    }
    else return false;
};
```

[summary]

- Algorithms are awesome
- Invest in yourself

[questions]