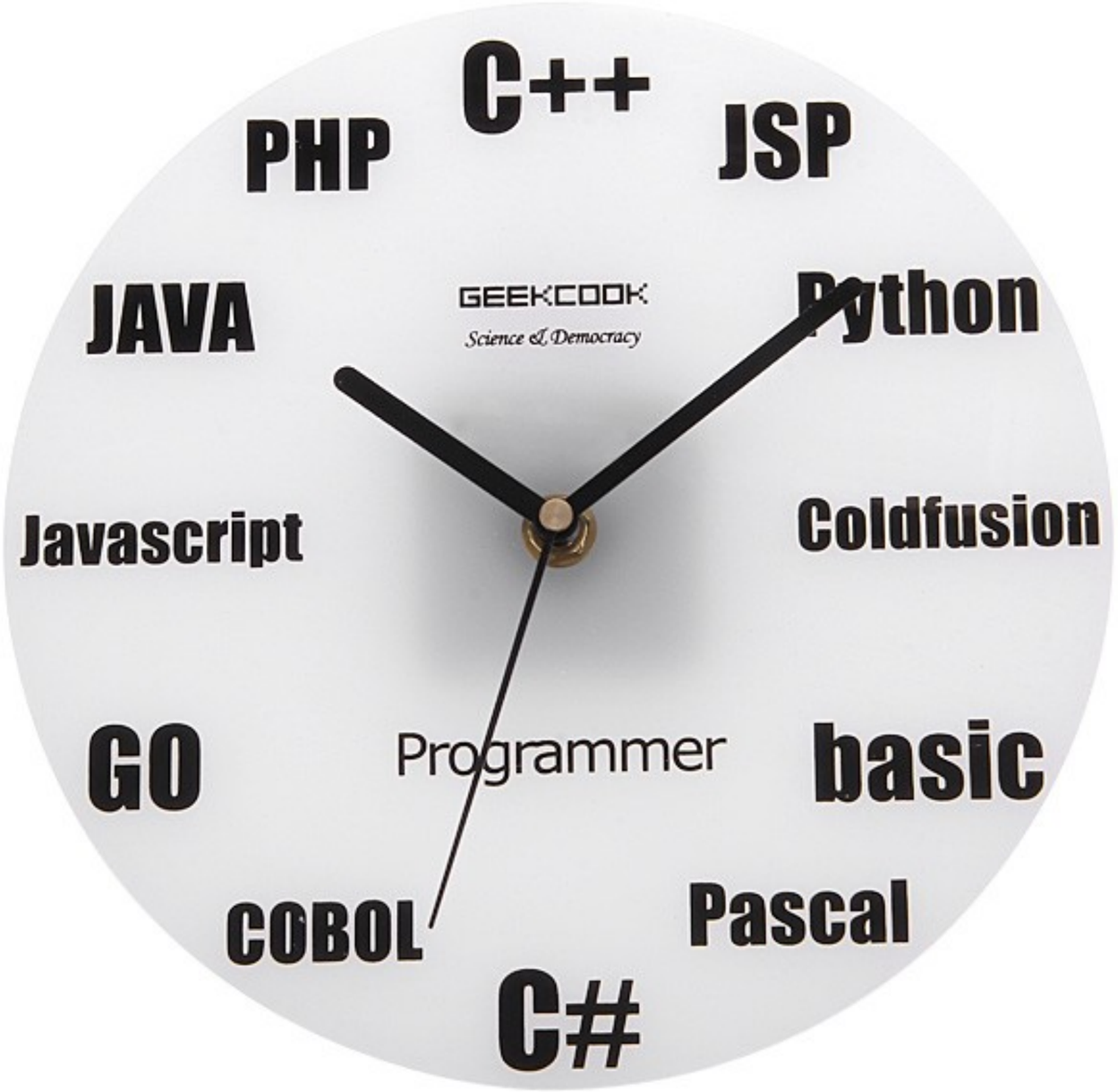
Objektinis Programavimas

Programos spartos matavimas



Turinys

Motyvacija (1)

— Viena iš labiausiai akivaizdžių ir reikalingų bibliotekų, kurią turi turėti programavimo kalbą yra skirta darbui su laiku.

— Tokią biblioteką suprojektuoti yra gana sudėtinga.

— Praeityje naudoti interfeisai (`C`, `POSIX`) keitėsi matavimo tikslumu: nuo sekundžių iki milisekundžių, tuomet iki mikrosekundžių ir galiausiai iki nanosekundžių.

Motyvacija (2)

— Bėda ta, kad kiekvienam atnaujinimui buvo sukurta vis nauja sąsaja (interfeisas).

— Dėl šios priežasties `C++11` standarte buvo pasiūlyta tikslumui neutrali biblioteka.

— Ši biblioteka paprastai vadinama `std::chrono` biblioteka__, nes jos funkcijos yra apibrėžtos `<chrono>` header'yje.

<chrono> vs. <ctime>

- <ctime> - yra C stiliaus antraštė, t.y. senas, "not type safe" ir ne toks tikslus kaip <chrono> matavimo būdas.
- <chrono> - "type safe, tikslesnis, bei turi išplėstinį funkcionalumą lyginant su <ctime>.
- Tačiau kartais visvien privalome vietoj <chrono> naudoti <ctime>:
  - kai susiduriame su C-API.
  - kai norime gauti datas iš laiko taškų (apibrėžta vėliau), nes net ir C++17 neturi tam tikro funkcionalumo.

`std::chrono` biblioteka

— `chrono` biblioteka sukurta atsižvelgiant į tai, kad laikrodžiai (clocks) gali skirtingose sistemose skirtis, o laikui bėgant tikslėti.

— Tikslumui neutrali koncepcija sukurta atskyrus trukmę (duration) ir laiko tašką ("timepoint") nuo konkrečių laikrodžių (clocks).

— Taigi, `std::chrono` biblioteka apibrėžia tris pagrindinius tipus (kartu su pagalbinėmis funkcijomis):

  — laikrodžiai (clocks)

  — laiko taškai (time points)

  — trukmės (durations)

Laikrodžiai (clocks**)**

Kas yra laikrodis (clock**)?**

Laikrodis susideda iš laiko pradžios taško (**epochos**) ir **ticks**'ų dažnumo. Pavyzdžiui, laikrodis gali turėti epochą - 1970 m. Sausio 1 d. (ši diena yra UNIX ir POSIX sistemų laikrodžių epocha) ir matuoti laiką sekundės dažnumu.

Laikrodžių (clocks**) tipai**

C++11 apibrėžia šiuos laikrodžių tipus:

— system_clock - "wall clock time from the system-wide realtime clock"

— steady_clock - "monotonic clock that will never be adjusted"

— highresolutionclock - "the clock with the shortest tick period available"

Skirtingų laikrodžių savybės[ls]

```
#include <chrono>
#include <iostream>
#include <iomanip>

template <typename C>
void printClockData () {
    using namespace std;
    cout << "- precision: ";
    // if time unit is less or equal one millisecond
    typedef typename C::period P;// type of time unit
    if (ratio_less_equal<P,milli>::value) {
       // convert to and print as milliseconds
       typedef typename ratio_multiply<P,kilo>::type TT;
       cout << fixed << double(TT::num)/TT::den
            << " milliseconds" << endl;
    } else { // print as seconds
        cout << fixed << double(P::num)/P::den << " seconds" << endl;
    }
    cout << "- is_steady: " << boolalpha << C::is_steady << endl;
}

int main() {
    std::cout << "system_clock: " << std::endl;
    printClockData<std::chrono::system_clock>();
    std::cout << "\nhigh_resolution_clock: " << std::endl;
    printClockData<std::chrono::high_resolution_clock>();
    std::cout << "\nsteady_clock: " << std::endl;
    printClockData<std::chrono::steady_clock>();
}
```

[ls] **5.7. Clocks and Timers** iš C++ Standard Library, The: A Tutorial and Reference, 2nd Edition

Laiko taškas (timepoint**) ir trukmė (**duration**)**

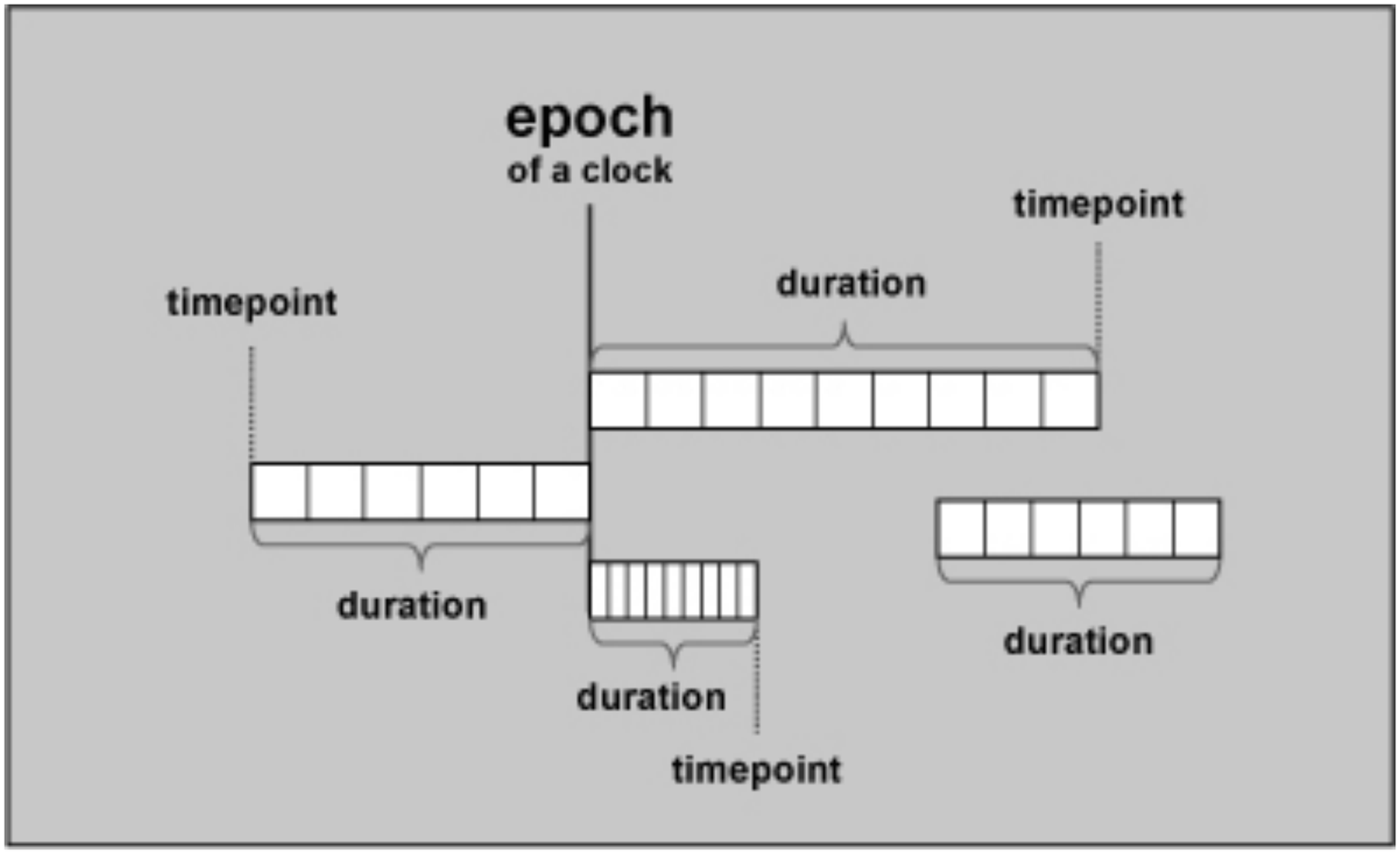
Kas yra laiko taškas (timepoint**)?**

Laiko taškas yra laiko **trukmė**, praėjusi nuo konkretaus laikrodžio (clocks) epochos. Tipiškas pavyzdys yra laiko taškas žymintis: "2000 metų vidurnaktį", kuris suprantamas kaip "1 262 300 400 sekundžių nuo 1970-01-01 dienos".

Kas yra trukmė (duration**)?**

Trukmė (duration) susideda iš laiko intervalo (tarp dviejų laiko taškų (timepoint'ų)), apibrėžiamo kaip tam tikro laiko vieneto **ticks**'ų skaičius per tam laiko vienetą . Pvz. "3 minučių trukmė" suprantama, kaip 3 "1-minutės" trukmės **ticks**'ai.

`std::chrono` **biblioteka (3)**



`std::chrono` panaudojimo pavyzdys (1)

```cpp
#include <iostream>
#include <chrono>
#include <vector>

int main() {
    auto start = std::chrono::high_resolution_clock::now();  // Paleisti
    std::vector<int> v;
    for (size_t i = 0; i < 10000000; ++i)
        v.push_back(i);
    auto end = std::chrono::high_resolution_clock::now();    // Stabdyti
    std::chrono::duration<double> diff = end-start;          // Skirtumas (s)
    std::cout << "10 000 000 elementų užpildymas užtruko: "
              << diff.count() << " s\n";
}
```

`std::chrono` naudojimo pavyzdys (2)

```cpp
#include <iostream>
#include <chrono>
#include <vector>

using namespace std::chrono; // sub-namespace std::chrono;

int main() {
    auto start = high_resolution_clock::now();  // Paleisti
    std::vector<int> v;
    for (size_t i = 0; i < 10000000; ++i)
        v.push_back(i);
    auto end = high_resolution_clock::now();    // Stabdyti
    duration<double> diff = end-start;          // Skirtumas (s)
    std::cout << "10 000 000 elementų užpildymas užtruko: "
              << diff.count() << " s\n";
    std::cout << "10 000 000 elementų užpildymas užtruko: "
              << duration_cast<milliseconds>(end-start).count() << " msec\n";
}
```

`std::chrono` apibendrinimas

— C++11 standarte atsiradusi laiko matavimui skirta `std::chrono` biblioteka yra labai funkcionali ir išsprendė ankstesniuose C++ standartuose buvusias problemas.

— Deja, bet `std::chrono` naudojimas nėra labai patogus ir reikalauja nemažai sunkiai atsimenamo (nors to šiais laikais tikrai nereikia daryti 😄) kodo rašyti.

Laiko matavimo klasė (1)

```
#include <chrono>

class Timer {
private:
    std::chrono::time_point<std::chrono::high_resolution_clock> start;
public:
    Timer() : start{std::chrono::high_resolution_clock::now()} {}
    void reset() {
        start = std::chrono::high_resolution_clock::now();
    }
    double elapsed() const {
        return std::chrono::duration<double>
        (std::chrono::high_resolution_clock::now() - start).count();
    }
};
```

Laiko klasės naudojimo pavyzdys

```
#include <iostream>
#include <chrono>
#include <vector>
#include "Timer.h"

int main() {
    Timer t;  // Paleisti
    std::vector<int> v;
    for (size_t i = 0; i < 10000000; ++i)
        v.push_back(i);
    std::cout << "10 000 000 elementų užpildymas užtruko: "
              << t.elapsed() << " s\n";
}
```

Laiko matavimo klasė (2)

```
#include <chrono>

class Timer {
private:
  // panaudojame using
  using hrClock = std::chrono::high_resolution_clock;
  using durationDouble = std::chrono::duration<double>;
  std::chrono::time_point<hrClock> start;
public:
    Timer() : start{ hrClock::now() } {}
    void reset() {
        start = hrClock::now();
    }
    double elapsed() const {
        return durationDouble (hrClock::now() - start).count();
    }
};
```

Komentarai dėl laiko matavimo

1. Įsitikinkite, kad sukompiliavote naudodame `release`, o ne `debug`, nes `debug` programa veikia gerokai lėčiau.
2. Naudokite tuos pačius kompiliatorių flag'us. Tokie flag'ai kaip `-O3` gali turėti didelę įtaką programos spartai. Daugiau info apie kompiliatorių flag'us: https://stackoverflow.com/a/1778700/3737891
3. Eksperimentą pakartokite keletą kartų, pvz. 5-10 ir imkite vidurkinius rezultatus.
4. Eksperimentų metu geriausia, kad kompiuteris nedarytų jokių kitų veiksmų, galinčių įtakoti laiko matavimus.

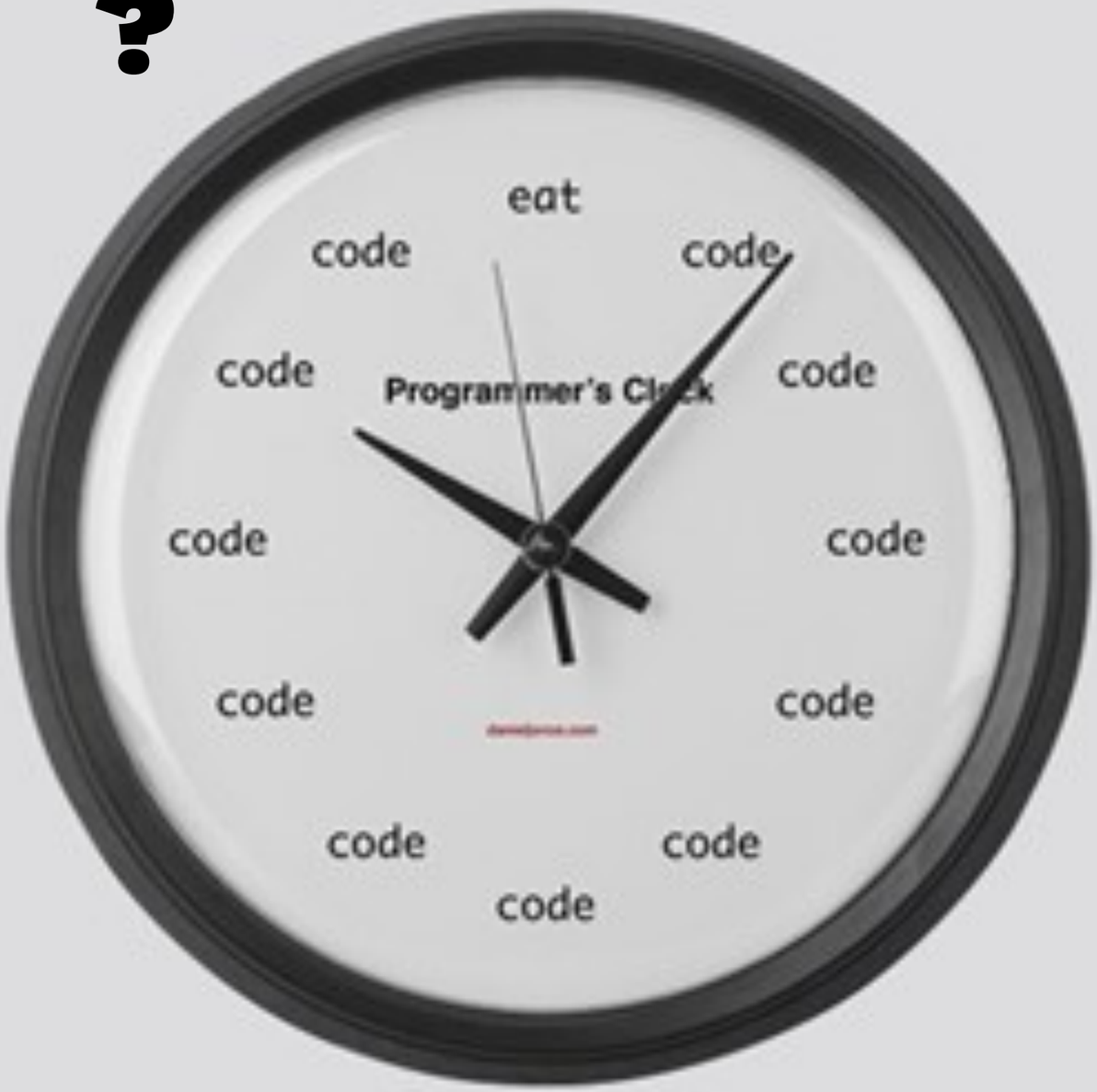
Naujos `std::chrono` **galimybės** `C++20` **standarte**

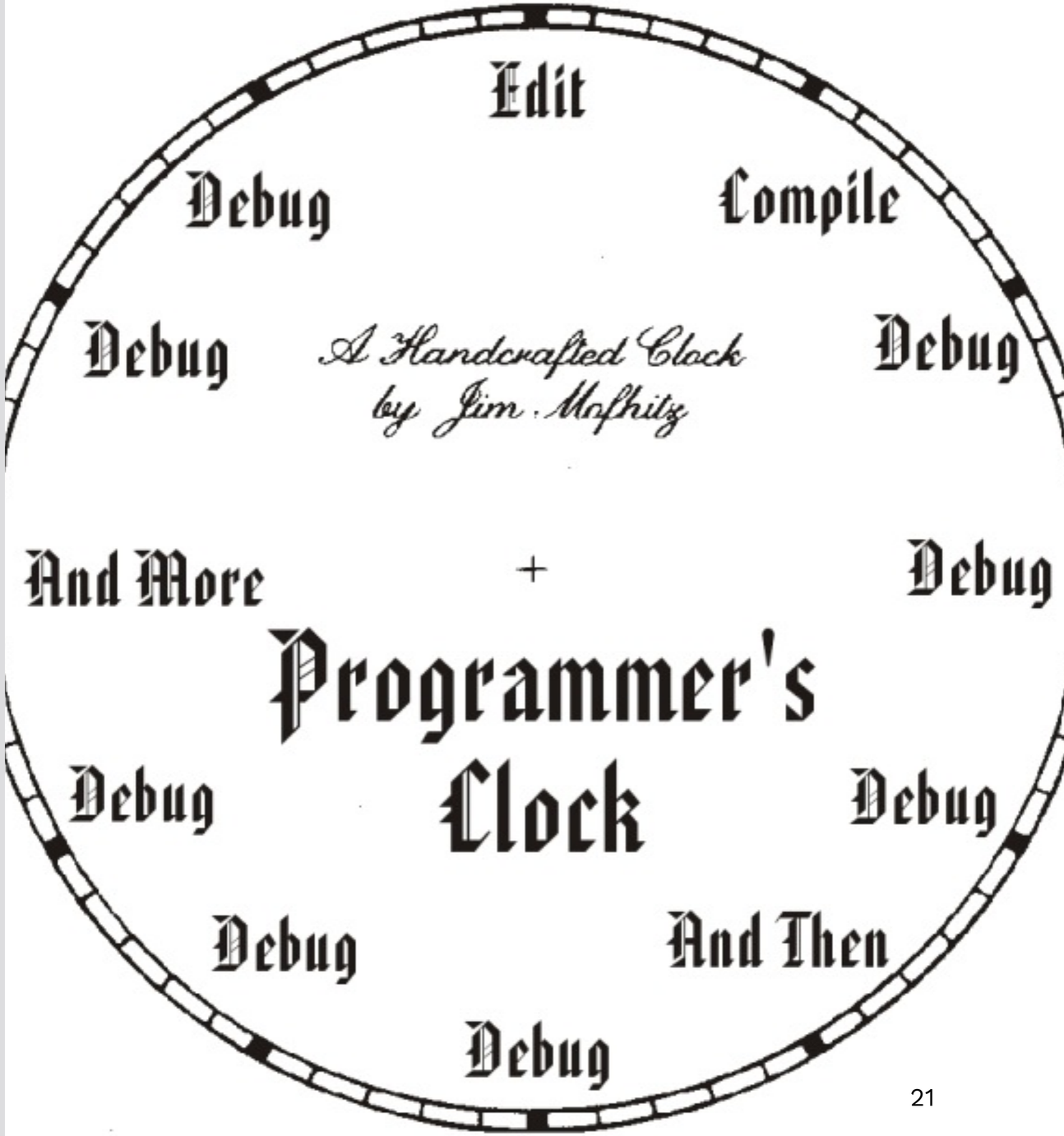
— `C++20` standarte pridedamas elegantiškas ir efektyvus <chrono> išplėtimas, palaikantis:
  — ilgesnius laiko intervalus (pvz., metus ir mėnesius)
  — kalendorius
  — laiko juostas.
— Daugiau info ir pavyzdžių: https://en.cppreference.com/w/cpp/chrono

Literatūra

1. **5.7. Clocks and Timers** iš C++ Standard Library, The: A Tutorial and Reference, 2nd Edition
2. https://en.cppreference.com/w/cpp/chrono

?


eat
code
code
code
code
code
code
code
code
code
code
code


Edit
Debug
Compile
Debug
A Handcrafted Clock
by Jim Mofhitz
Debug
And More
+
Debug
Programmer's
Clock
Debug
Debug
Debug
And Then
Debug