2023 12 22

```cpp
// std::chrono:duration
using namespace std;
using namespace std::chrono;

// count()
int main()
{
    milliseconds ms1 { 321313 };

    auto val = ms1.count(); //  321313
}
```

```cpp
// duration fonksiyonlar
int main()
{
    // nanoseconds
    auto dr = milliseconds{456}  + nanoseconds{6'423'123} + seconds { 2 };
    cout << dr.count();

    milliseconds ms {2131};
    ms = 45; // örtülü dönüşüm syntax hatası
    int ival = ms; // syntax hatası
}
```

```cpp
int main()
{
    milliseconds ms {3123};
    microseconds us {4'434'543};

    us = ms; // geçerli
    ms = us; // geçersiz (daha ince türden daha kaba türe dönüşüm syntax hatası
olr)
}
```

```cpp
using halfsec = duration<double, ratio<1, 2>>;
 int main()
 {
     halfsec hs = ms; // geçerli
 }
```

```cpp
int main()
 {
     cout << milliseconds{ 345 } << "\n"; // 345ms --C++20 ile geldi
 }
```

UDL (user-defined literals)

```
// operator""ms(823) --823ms operator overloading
 int main()
 {
     auto dur = 345ms;
     constexpr auto dur = 345ms;
     constexpr auto dur1 = operator""ms(345);

     auto val = 567ms + 3457us + 65423ns;
 }
```

```
int main()
 {
     "mustafa"s // string sınıfından
     operator""s("alican", 6); // "alican"s
 }
```

```
    operator""_x(long long double);
    operator""_y(unsigned long long);
    operator""_z(char);
    operator""_k(const char*, size_t);

    cooked   569s 569'u doğrudan operator fonksiyonuna gönderiyorum
    uncooked  21312x bir yazı olarak gönderiyorsak null karakter görene kadar
okuyosak
```

```
// UDL oluşturma (namespace almak daha mantıklı)
constexpr double operator""_m(long double val)
{
     return static_cast<double>(val);
}

constexpr double operator""_cm(long double val)
{
     return static_cast<double>(val / 100);
}

constexpr double operator""_mm(long double val)
{
     return static_cast<double>(val / 1000);
}

constexpr double operator""_km(long double val)
{
     return static_cast<double>(val * 1000);
}
int main()
{
     constexpr auto x = 4.5_m // double
     constexpr double y =  operator""_m(4.5657);

     constexpr auto distance = 3.5_m + 876.35_cm + 1234.12_mm + 0.0812_km;

     cout << distance << " metre\n";
}
```

```cpp
constexpr std::size_t operator""_KB(unsigned long long size)
{
    return static_cast<std::size_t>(size * 1'024);
}
constexpr std::size_t operator""_MB(unsigned long long size)
{
    return static_cast<std::size_t>(size * 1'024 * 1'024);
}
constexpr std::size_t operator""_GB(unsigned long long size)
{
    return static_cast<std::size_t>(size * 1'024 * 1'024 * 1'024);
}

int main()
{
    std::array<char, 12_KB> buf1{};
    std::array<char, 1_MB> buf2{};
}
```

```cpp
constexpr int operator""_i(char c)
{
    return c;
}
int main()
{
    using namespace std;

    cout << 'A' << "\n"; // A
    cout << 'A'_i << "\n"; // 65

}
```

```cpp
unsigned constexpr int operator""_b(const char* p)
{
    using result{};
    while (*p)
    {
        char digit = *p;

        if (digit != '1' && digit != '0')
        {
            throw std::runtime_error{"invalid ch : "s + digit}İ
        }

        result = result * 2 (digit - '0');
        ++p;
    }

    return result;
}
int main()
{
    using namespace std;

    auto val = 1010101_b;
    cout << val << "\n";
}
```

```cpp
class Kilogram
{
    public:
        class prevent_usage{};
        Kilogram(prevent_usage, double val) : mweigth{ val } {}
        friend constexpr Kilogram operator+(const Kilogram& x, const Kilogram& y)
        {
            return Kilogram{Kilogram::prevent_usage{}, x.mweigth + y.mweigth};
        }
    private:
        double mweigth;
};

constexpr Kilogram operator""_kg(long double x)
{
    return Kilogram{ Kilogram::prevent_usage{}, static_cast<double>(x)};
}

constexpr Kilogram operator""_gr(long double x)
{
    return Kilogram{ Kilogram::prevent_usage{}, static_cast<double>(x / 1000)};
}

constexpr Kilogram operator""_mg(long double x)
{
    return Kilogram{ Kilogram::prevent_usage{}, static_cast<double>(x / 1000 /
1000)};
}


int main()
{
    constexpr Kilogram kg1  = 5.6_kg;
    constexpr auto weigth = 4.5_kg + 1234.87_gr + 345'434.55_mg;
}
```

```cpp
// std::chrono  (duration_cast)
int main()
{
    using namespace std::chrono;

    auto ms == 75624ms;
    seconds s = ms; // sytantx hataso

    seconds s(ms.count() / 1000); // geçerli

    duration_cast<seconds>(ms); // veri kaybı olur
    cout << sec.count() << "\n"; // 75
}
```

```cpp
int main()
{
    long long val;
    std::cout << "milisaniye olarak degeri girin : ";
    cin >> val;

    milliseconds ms{val};

    hours hrs = duration_cast<hours>(ms);
    minutes mins = duration_cast<minutes>(ms % 1h);
    seconds sec = duration_cast<seconds>(ms % 1min);

    if (hrs.count())
        cout << hrs.count() << " saat";
    if (mins.count())
        cout << mins.count() << " dakika";
    if (sec.count())
        cout << sec.count() << "mili saniye"\n";
}
```

```cpp
// yuvarlama işlemleri
int main()
{
    auto ms = 923'879ms;

    cout << duration_cast<seconds>(ms) << "\n"; // 923
    cout << floor<seconds>(ms) << "\n"; // 923
    cout << ceil<seconds>(ms) << "\n"; // 924
    cout << round<seconds>(ms) << "\n"; // 924
}
```

```cpp
template <typename Rep, typename Period>
std::ostream& operator<<(std::ostream& os, const std::chrono::duration<Rep,
Period>& dur)
{
    return os << dur.count() << " * " << Period::num << " / "  << Period::den <<
}

using tensec = std::chrono::duration<int, std::ratio<10>>
int main()
{
    using namespace std;
    using namespace std::chrono;

    cout << 3456ms; // 3456 * ( 1 / 1000)
    cout << tensec {6512} << "\n";
}
```

```cpp
// system_clock
int main()
{
    using namespace std;
    using namespace std::chrono;

    system_clock::time_point // time_point<system_clock, system_clock::duration>

    auto tp1 = system_clock::now();

    auto tp2 = system_clock::now();
    auto diff = tp2 - tp1;
}
```

```cpp
// süre ölçümü
std::vector<size_t> get_sorted_vector(std::size_t n)
{
    std::vector<size_t > vec(n);

    std::generate_n(vec.begin(), n, rand);
    std::sort(vec.begin(), vec.end());

    return vec;
}
```

```cpp
int main()
{
    using namespace std;
    using namespace std::chrono;
    auto tp_start = steady_clock::now();
    auto vec = get_sorted_vector(450'000);

    auto tp_end = stready_clock::now();

    cout << duration_cast<milliseconds>(tp_end - tp_start).count() << "
milisaniye\n";
    cout << duration_cast<double>(tp_end - tp_start).count() << " saniye\n";
    cout << duration_cast<double, micro>(tp_end - tp_start).count() << " micro
saniye\n";

    cout << "vec.size()" << vec.size() << "\n";
}
```

```cpp
int main()
{
    using namespace std;
    using namespace std::chrono;

    auto tp = system_clock::now();

    time t ct_time == system_clock::to_time_t(tp);
    cout << ctime(&c_time) << "\n";
}
```