2023 12 25

```cpp
//system_clock

#include <chrono>
#include <ctime>
int main()
{
    using namespace std;
    using namespace std::chrono;

    days oneday{ 1 };

    auto tp_today = system_clock::now();

    auto tp_tomorrow = tp_today + oneday;

    time_t t1 = system_clock::to_time_t(tp_today);
    time_t t2 = system_clock::to_time_t(tp_tomorrow);

    cout << ctime(&t1); << "\n"; // şu an
    cout << ctime(&t2); << "\n"; // bir gün sonrası
}
```

vocabulary type (C++ 17)

```
        std::optional   --template class
            bir değerin olması ya da olmaması problem domaininde söz konusu ise
kullanılır

        std::variant    --template class
             n tane farklı türden herhangi bir değer tutabilir
        std::any        --class
            void* türüne  alternatif
```

std::optional

```cpp
// std::optional
#include <optional> // wrapper
#include <string>

int main()
{
    using namespace std;

    optional<string> o1;
    optional<int> o2;
    optional<Date> o3;
}
```

```cpp
#include <optional> // wrapper
#include <string>

int main()
{
    using namespace std;
    optional<int> x{ 678 };

    if (x) // x.operator bool()
        std::cout << "dolu bir degere sahip\n";
    else
        std::cout << "bos, bir degere sahip degil\n";

    optional<int> x1{ nullopt }; // constexpr bir değişken boş halde ayağa gelir

    optional x2 = 10; // CTAD  optional<int>
    optional x3 = "damla"; // optional<const char*>
    optional x3 = "damla"s; // optional<std::string>
}
```

```cpp
int main()
{
    optional<int> x1; // boş
    optional<int> x2{}; // boş
    optional<int> x3{nullopt}; // boş
    optional<int> x4(); // fonksiyon bildirimi bu

    if (x1.has_value()) // false
}
```

```cpp
int main()
{
    optional x(4345);

    cout << *x << "\n"; // 4345
    *x = 123;
    cout << *x << "\n"; //123

    optional<string> osx{ "naci cemal" };
    cout << *osx << "\n";
    osx->length();

    *osx+= " dagdelen";
    // eğer optional boş ise * -> operatorlarını kullanmak tanımsız davranıştır.
    cout << osx.value() << "\n";
}
```

```cpp
// value_or()
int main()
{
    // eğer boşsa değeri geçer
    optional<string> os{ "furkan mert" };
    cout << os.value_or("noname"); << "\n"; // furkan mert
    os = {};
    cout << os.value_or("noname"); << "\n"; // noname
}
```

```cpp
//std::optional Kullanım
void print_e_mail(const std::optional<std::string>& op)
{
    std::cout << "e-posta adresi: " << op.value_or("yok") << "\n";
}

int main()
{
    print_e_mail("necati ergin");
    print_e_mail(std::nullopt);
}
```

```cpp
class Person
{
    private:
        //...
        std::optional<std::string> m_middle_name;
};
```

```cpp
class Record {
    public:
        Record(const std::string& name, std::optional<string> nick,
std::optional<int> age) :
        m_name(name), m_nick(nick), m_age(age) {}

    private:
        std::string m_name;
        std::optional<std::string> m_nick;
        std::optional<int> m_age;

};

int main()
{
    Record r1{"emre bahtyiar", nullopt, nullopt};
}
```

```cpp
std::optional<int> to_int(const std::string& s)
{
    try
    {
        return stoi(s);
    }
    catch(...)
    {
        return std::nullopt;
    }
}
```

```cpp
class A
{
    public:
        A() {};
        A(int) {};
};

int main() {

    using namespace std;
    optional<A> ox;

    ox.emplace(); // default ctor
    // yeni bir emplace'te önce dtor eder sonra yenisini oluşturur
    ox.emplace(2); // int ctor
    // sonrasında sarmaladığı nesneyi sonlandırır

    ox = 5; // legal implictly
}
```

```cpp
// in_place
#include "date.h"
int main()
{
    using namespace std;

    //optional<Date> ox(4, 5 1987); // hatalı
    optional<Date> ox(in_place, 4, 5, 1987) // gecerli
}
```

```cpp
int main()
{
    using namespace std;
    optional<complex<double>> ox{in_place, 2.3, 6. 7};
}
```

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

    optional<string> o1(in_place, 20, 'X');
    auto o2 = make_optional<string>(10, 'Y');
}
```

```cpp
// optional karşılaştırma
int main()
{
    using namespace std;

    optional x = 5;
    optional y = 125;

    cout << (x == y) << "\n";
    cout << (x != y) << "\n";
    cout << (x < y) << "\n";

    //bos olursa  her zaman küçük
    optional<int> z;
    cout << x < z << "\n"; // true
    cout << x < y << "\n"; // true

}
```

std::variant

```cpp
// std::variant
#include<variant>
// union ile benzer
union Nec {
    int x;
    double dval;
};

int main()
{
    // x'in türü ilk tanımlanan yani int olacak
    variant<int, double, long, char> x; // x'ın türü bunlardan biri olacak
    cout << x.index() << "\n"; // 0

    variant<int, double, long, char> x(4.5);
    cout << x.index() << "\n"; // 1 yani double
}
```

```cpp
int main()
{

    variant<int, double, long, char> x;
    cout << x.index() << "\n"; // 0

    x = 3.4;
    cout << x.index() << "\n"; // 1

    x = 'a';
    cout << x.index() << "\n"; // 3
}
```

```cpp
//holds_alternative
int main()
{
    variant<string, double, int> x{57};

    cout << "x.index() = " << x.index() << "\n";

    if (holds_alternative<int>(x))
    {
        cout << "int tutuyor" << "\n";
    }
}
```

```cpp
// get<>
int main()
{
    variant<int, double, long> x;
    // value init edilir yani garabe value değil
    cout << get<0>(x) << "\n"; // 0
    x = 3.7;
    cout << get<double>(x) << "\n"; //

    get<3>(x); // geçersiz indeks syntax hatası
}
```

```cpp
int main()
{
    variant<int, double, long> x{3.6};

    try
    {
        autu val = get<2>(ex) //
    }
    catch (const std::exception& ex)
    {
        // bad variant acces throw eder
        std::cout << "exception caught: " << ex.what() << "\n";
    }
}
```

```cpp
// in_place_index ve in_place_type
int main()
{
    variant<int, Date, string> x{in_place_index<1>, 12, 5, 1987};
    variant<int, Date, string> x1{in_place_type<Date>, 2, 5 1987};

    cout << get<1>(x) << "\n";

    x.emplace<string>(10, 'a');
    cout << get<2>(x) << "\n";
}
```

```cpp
// monostate
int main()
{
    using namespace std;
    // hiçbir değer tutmuyor monostate tutuyor boş gibi
    variant<monostate, int, double, long> x;
}
```

```cpp
// get_if
int main()
{
    using namespace std;

    variant<int, double, long> x(2.3);
    // eğer int'se adress döndürür değilse nullptr
    if (auto *ptr = get_if<int>(&x))
    {
        cout << *ptr << "\n";
    }
}
```