# 2024 01 03

### std::random

```cpp
// std::bernoulli_distribution
#include <random>
#include <iostream>
#include <chrono>
int main()
{
    using namespace std;
    // exe her çalıştığın da farklı bir seed(tohum) çalışacak
    mt19937 eng{std::chrono::steady_clock::now().time_since_epoch()};
    mt19937 eng1{ random_device{}() };
    bernoulli_distribution dist{0.81}

    std::size_t n = 1'000'000u;
    int cnt{};

    for (size_t i{}; i < n; ++i)
    {
        if (dist(eng))
        {
            ++cnt;
        }
    }

    cout << static_cast<double>(cnt)  / n <<  "\n";
}
```

```cpp
// her çağrıldığında aynı engine çağrılacak
std::mt19937& engine()
{
    static std::mt19937 eng{ std::random_device{}() };
    return eng;
}

int main()
{
    using namespace std;
    uniform_int_distribution dist{0 , 99};

    cout << dist(engine()) << "\n";
}
```

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

    uniform_int_distribution dist1{ 0, 99 };

    auto prm = dist1.param();
    //dist1 parametrelerini dist2'ye geçirdik
    uniform_int_distribution dist2{dist1.param()}; //
}
```

```cpp
/// algoritmalarda random
int main()
{
    using namespace std;
    vector<int> ivec(100'000);

    mt19937 eng;
    uniform_int_distribution dist{ 4671, 8812};
    generate(ivec.begin(), ivec.end(), [&eng, &dist]{ return dist(eng);});
}
```

## Concurrency

```cpp
#include <thread>
int main()
{
    using namespace std;
    // move only type
    thread th1{[] {cout << "emre bahtiyar\n"}}; // ctor'u callable alır
    thread th1{[] {cout << "emre bahtiyar\n"}}; // ctor'u callable alır
    thread th2{[] {cout << "emre bahtiyar\n"}}; // ctor'u callable alır
    thread th3{[] {cout << "emre bahtiyar\n"}}; // ctor'u callable alır
}
```

```cpp
void func(int a, int b, int c)
{

}

int main()
{
    using namespace std;

    // eğer join() ya da detach() fonksiyonu çağrılmaz terminate fonksiyonu
çağrılır
    thread tx{ func, 10, 20, 30 };

    // fonksiyon bitene kadar bu nokta beklesin anlamına gelir
    tx.join();
    // fonksiyon ile ana thread ayrılır tx kendi biter
    tx.detach();
}
```

```cpp
/*
    Bir thread nesnesi joinable ve unjoinable durumunda olabilir,
    eğer bir thread nesnesi joinable durumduysa ve dtor'u çağrılırsa
    doğrudan terminate fonksiyonu çağrılır.
*/

void nec_terminate()
{
    std::cout << "ne terminate cagrildi\n";
    std::abort();
}

void func()
{
    std::cout << "ben func fonksiyonuyum\n";
}

int main()
{
    using namespace std;

    set_terminate(nec_terminate);

    {
        thread tx{func};
        tx.join(); // bunu çağırmazsak terminate çağrılacak
    }
}
```

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

    thread t;
    cout << t.joinable() << "\n"; // false çünkü iş yükü yok

    thread t1{[] {}};
    cout << t1.joinable() << "\n"; // true çünkü iş yükü var
    t1.join();
    cout << t.joinable() << "\n"; // false artık join edilmiş
}
```

```cpp
// thread taşıma
int main()
{
    thread t1{[]{}};
    thread t2{move(t1};

    cout << t1.joinable() << "\n"; // false artık taşındı.
    cout << t2.joinable() << "\n"; // true.

    t2.join();

    cout << t1.joinable() << "\n"; // false
    cout << t2.joinable() << "\n"; // false.
}
```

```cpp
void func()
{
    std::cout << "ben func fonksiyonuyum\n";
}

int main()
{
    using namespace std;

    thread tx{ func };
    tx.join();

    tx.join(); // exception throw eder
}
```

```cpp
// jthread --cpp 20

void func()
{
    throw std::runtime_error{ " hataaaa" };
}

void foo()
{
    //code
}

void bar()
{
    /*
        dtor çağrıldığında jthread eğer sarmaladğı thread'in joinable ise join
eder

    */
    jthread t{ foo };

    func(); // exception throw eder
    t.join(); // buraya girmez
}

int main()
{
    try
    {
        bar();
    }
    catch (std::exception &ex)
    {

    }
}
```

```cpp
void func(char c)
{
    for (int i = 0; i < 1000; ++i)
    {
        std::cout.put(c);
    }
}

int main()
{
    using namespace std;

    vector<thread> tvec(26);

    for (auto& t : tvec)
    {
        t = thread{func, c++}; // taşınma semantiği
    }

    for (auto& t : tvec)
    {
        t.join();
    }
}
```

thread'e verdiğimiz callable exception verirse exception'ı yakalamayız

```cpp
int foo(int x, int y)
{
    // code
    return x * y;
}

int main()
{
    using namespace std;

    thread tx{ foo, 10, 56 };
    // foo fonksiyonun geri dönüş değerini thread nesnesi aracıyla alamayız
    tx.join();
}
```

```cpp
// thread fabrika fonksiyonu
std::thread make_thread()
{
    std::thread tx{[]
    {
        std::cout << "emre\n";
    }};
    // otomatik ömürlü nesneler l value to x value olr
    return tx;
}

std::thread func(std::thread tx)
{
    return tx;
}

int main()
{
    using namespace std;

    auto t1 = make_thread();
    auto t2 = move(t1);
    t1 = func(move(t2));

    t1.join();
}
```

```cpp
// threadlere verilebilecek callable'lar
class Functor
{
    public:
        void operator()(int x)const{
            std::cout << x << "\n";
        }
}
void func(int x)
{
    std::cout << x << "\n";
}
struct Nec
{
    static void print(int x)
    {
        std::cout << x;
    }
    void display(int x)const
    {
        std::cout << x;
    }
}
int main()
{
    using namespace std;

    thread t1(func, 1); // fonksiyon
    thread t2([]{int x} {cout << x << "\n";}, 2); // lambda
    thread t3(Functor{}, 3);
    thread t4(Nec::print, 4);

    Nec mynec;
    thread t5(&Nec::display, mynec, 5);
    // çıktı belli olmaz
    t1.join();
    t2.join();
    t3.join();
    t4.join();
    t5.join();
}
```

```cpp
void func(int& r)
{
    r +=200;
}

int main()
{
    using namespace std;
    int x = 67;

    thread t{ func, x}; // syntax hatası
    cout << "x = " << x << "\n";

    thread tx{ func, ref(x)}; // syntax hatası yok;
    cout << "x = " << x << "\n"; // 267
}
```

```cpp
void func(std::string && r)
{
    std::cout << r.size() << "\n";
    auto x = std::move(r);
}

int main()
{
    using namespace std;

    string name{ "emre bahtiyar "};

    thread tx{ func, name};
    tx.join();

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