

Objektinis Programavimas

**Atsitiktinių skaičių
generavimas**



Turinys

1. Motyvacija
2. rand() ir srand() funkcijos C/C++
3. rand() Considered Harmful
 - rand() ir srand() funkcijų trūkumai
4. Atsitiktinių skaičių generavimas su <random>
5. Ką daryti jei Jūsų sistema neturi random_device?
6. Klasė skirta atsitiktinių skaičių generavimui

Motyvacija

- Viena iš labiausiai akivaizdžių ir reikalingų bibliotekų, kurią turi turėti programavimo kalbą yra skirta atsitiktinių skaičių generavimui.
- Su atsitiktiniais skaičiais susiduriame įvairiose srityse ir situacijose, kaip pvz., testavime, žaidimų kūrime, modeliavime, kriptografijoje ir t.t.
- Bibliotekų, skirtų atsitiktinių skaičių generavimas sudarymas yra daug sudėtingesnis negu gali pasirodyti.
- Vis dar dažnai naudojamos C bibliotekos, kurios neužtikrina skaičių atsitiktinumo ir kitų būtinų charakteristikų.

```
int getRandomNumber()  
{  
    return 4; // chosen by fair dice roll.  
              // guaranteed to be random.  
}
```

`rand()` funkcija C/C++

- C kalboje atsitiktinių skaičių generavimui naudojama `rand()`.
- Kai generuojame atsitiktinių skaičių seką vient tik su `rand()` funkcija, tai kiekvieną kartą gausime tą pačią seką, kuomet tik programa bus vykdoma.

`int rand(void)`

returns a pseudo-random number in the range of 0 to `RAND_MAX`.
`RAND_MAX`: is a constant whose default value may vary between implementations but it is granted to be at least 32767.

rand() naudojimo pavyzdys

```
// C programa atsitiktinių skaičių generavimui
#include <stdio.h>
#include <stdlib.h>

int main() {
    // Kiekviena karta generuojama ta pati skaičių seka
    for(int i = 0; i < 10; ++i)
        printf(" %d ", rand());
    return 0;
}
```

srand() funkcija C/C++

- `srand()` nustato "pradžios tašką" (angl. **seed**) pseudo atsitiktinių (sveikųjų) skaičių serijai su `rand()` generuoti.
- Jei `srand()` praleidžiama (kaip buvo ankstesnio pvz. atveju), `rand()` `seed`'as interpretuojamas kaip konstanta lygi - `srand(1)`.
- Skirtingos `seed`'ų reikšmės priverčia `rand()` generuoti skirtingus atsitiktinius skaičius.

```
void srand( unsigned seed )
```

Seeds the pseudo-random number generator used by `rand()` with the value `seed`.

srand() naudojimo pavyzdys

```
// C programa atsitiktinių skaičių generavimui
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {

    // Naudoti dabartinį laiką kaip seed'a
    srand(time(0));

    // "Kiekvieną" kartą generuojama vis kita skaičių seka
    for(int i = 0; i < 10; ++i)
        printf(" %d ", rand());
    return 0;
}
```

rand() Considered Harmful^{rand}

- When you need a random number, don't call rand() and especially don't say rand() % 100! This presentation will explain why that's so terrible, and how C++11's <random> header can make your life so much easier.

rand() Considered Harmful

Stephan T. Lavavej ("Steh-fin Lah-wah-wade")
Senior Developer - Visual C++ Libraries
stl@microsoft.com

^{rand} <https://channel9.msdn.com/Events/GoingNative/2013/rand-Considered-Harmful>

What's Right With This Code?

```
#include <stdio.h>
```

All required headers are included!

```
#include <stdlib.h>
```

All included headers are required!

```
#include <time.h>
```

Headers are sorted!

```
int main() {
```

One True Brace Style!

```
    srand(time(NULL));
```

```
    for (int i = 0; i < 16; ++i) {
```

```
        printf("%d ", rand() % 100);
```

```
    }
```

```
    printf("\n");
```

%d is correct for int!

```
}
```

Unnecessary argc, argv, return 0; omitted!

What's **Wrong** With This Code?

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main() {
    srand(time(NULL));
    for (int i = 0; i < 16; ++i) {
        printf("%d ", rand() % 100);
    }
    printf("\n");
}
```

ABOMINATION!

Frequency: 1 Hz!

32-bit seed!

warning C4244:
'argument' :
conversion from
'time_t' to
'unsigned int',
possible loss
of data

Non-uniform distribution!

Range: [0, 32767]

Linear congruential → low quality!

Modulo → Non-Uniform Distribution

```
int src = rand(); // Assume uniform [0, 32767]
int dst = src % 100; // Non-uniform [0, 99]

//           [0, 99] src → [0, 99] dst
//           [100, 199] src → [0, 99] dst
// ...
// [32700, 32767] src → [0, 67] dst
```

- This is modulo's fault, not rand()'s
 - Trigger: input range isn't exact multiple of output range

Kaip išvengti % problemų?^{rand}

```
// Modifikuotas pavyzdys išvengiant `%` trūkumų
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main() {
    srand(time(NULL));
    int rand_sk;
    for (int i = 0; i < 16; ++i) {
        // invariantas: generuok tol, kol gausi rand_sk < "32700"
        // pagal skaidres, kur RAND_MAX = 32767
        do {
            rand_sk = rand();
        } while ( rand_sk > (RAND_MAX - (RAND_MAX % 100)) );
        printf("%d ", rand_sk % 100);
    }
    printf("\n");
}
```

^{rand} <https://channel9.msdn.com/Events/GoingNative/2013/rand-Considered-Harmful>

Atsitiktinių skaičių generavimas su <random>

- Taikomųjų sričių įvairovė atspindi platų atsitiktinių skaičių generatorių, kuriuos pateikia standartinė biblioteka <random>.
- Atsitiktinių skaičių generatorius susideda iš dviejų dalių:
 - Variklis (angl. **engine**), kuris generuoja atsitiktinių ar pseudo-atsitiktinių skaičių sekas.
 - Paskirstymas (angl. **distribution**), kuris sugeneruotas reikšmes transformuoja pagal matematinį pasiskirstymą norimame diapazone (intervale).

<random> URNGs

(Uniform Random Number Generators)

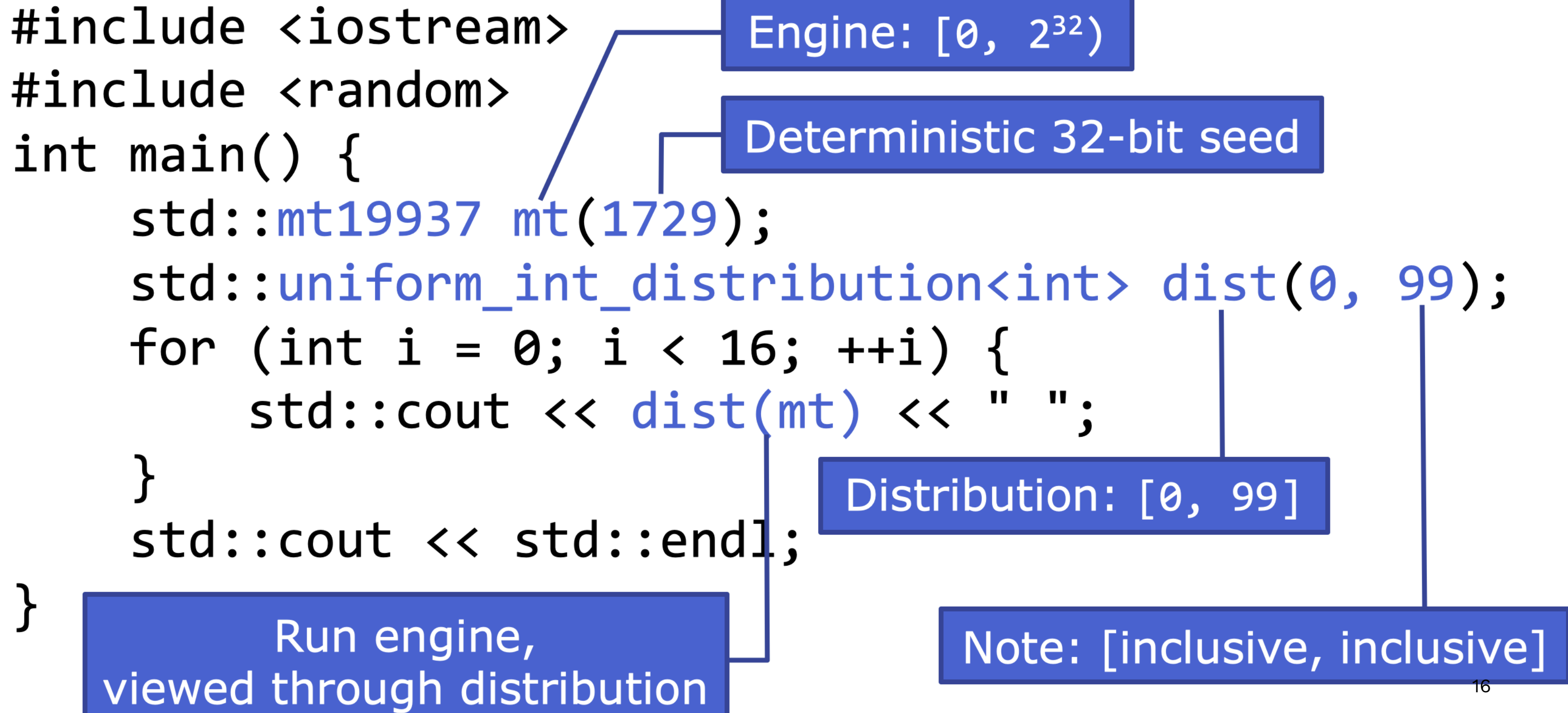
- Engine templates:
 - linear_congruential_engine
 - mersenne_twister_engine
 - subtract_with_carry_engine
- Engine adaptor templates:
 - discard_block_engine
 - independent_bits_engine
 - shuffle_order_engine
- Non-deterministic:
 - random_device
- Engine (adaptor) typedefs:
 - minstd_rand0
 - minstd_rand
 - mt19937
 - mt19937_64
 - ranlux24_base
 - ranlux48_base
 - ranlux24
 - ranlux48
 - knuth_b
 - default_random_engine

<random> Distributions

- Uniform distributions
 - `uniform_int_distribution`
 - `uniform_real_distribution`
- Poisson distributions
 - `poisson_distribution`
 - `exponential_distribution`
 - `gamma_distribution`
 - `weibull_distribution`
 - `extreme_value_distribution`
- Sampling distributions
 - `discrete_distribution`
 - `piecewise_constant_distribution`
 - `piecewise_linear_distribution`
- Bernoulli distributions
 - `bernoulli_distribution`
 - `binomial_distribution`
 - `geometric_distribution`
 - `negative_binomial_distribution`
- Normal distributions
 - `normal_distribution`
 - `lognormal_distribution`
 - `chi_squared_distribution`
 - `cauchy_distribution`
 - `fisher_f_distribution`
 - `student_t_distribution`

Hello, "Random" World!

```
#include <iostream>
#include <random>
int main() {
    std::mt19937 mt(1729);
    std::uniform_int_distribution<int> dist(0, 99);
    for (int i = 0; i < 16; ++i) {
        std::cout << dist(mt) << " ";
    }
    std::cout << std::endl;
}
```



Engine: $[0, 2^{32})$

Deterministic 32-bit seed

Distribution: $[0, 99]$

Note: [inclusive, inclusive]

Run engine, viewed through distribution

Hello, Random World!

```
#include <iostream>
#include <random>
int main() {
    std::random_device rd;
    std::mt19937 mt(rd());
    std::uniform_int_distribution<int> dist(0, 99);
    for (int i = 0; i < 16; ++i) {
        std::cout << dist(mt) << " ";
    }
    std::cout << std::endl;
}
```

Non-deterministic 32-bit seed

mt19937 vs. random_device

- mt19937 is:
 - Fast (499 MB/s = 6.5 cycles/byte for me)
 - Extremely high quality, but **not** cryptographically secure
 - Seedable (with more than 32 bits if you want)
 - Reproducible (Standard-mandated algorithm)
- random_device is:
 - Possibly slow (1.93 MB/s = 1683 cycles/byte for me)
 - Strongly platform-dependent (GCC 4.8 can use IVB RDRAND)
 - Possibly crypto-secure (check documentation, true for VC)
 - Non-seedable, non-reproducible

Ką daryti jei Jūsų sistema neturi random_device?

- Ankstesnėje skaidrėje matėme, kad `time(0)` ar `time(NULL)` panaudojimas atsitiktinių skaičių variklyje yra netoleruotinas, todėl kad `time(0)` kinta tik vienos sekundės dažnumu.
- Norėdami gauti geresnius seed'us, galime panaudoti `std::chrono` biblioteką:

```
#include <iostream>
#include <random>
#include <chrono>

int main() {
    using hrClock = std::chrono::high_resolution_clock;
    std::mt19937 mt(static_cast<long unsigned int>(hrClock::now().time_since_epoch().count()));
    std::uniform_int_distribution<int> dist(0, 99);
    for (int i = 0; i < 16; ++i) {
        std::cout << dist(mt) << " ";
    }
    std::cout << std::endl;
}
```

Klasė skirta atsitiktinių skaičių generavimui

```
class RandInt {  
public:  
    RandInt(int low, int high) : mt{rd()}, dist{low, high} { }  
    int operator()() { return dist(mt); } // generuok int'ą  
private:  
    std::random_device rd;  
    std::mt19937 mt;  
    std::uniform_int_distribution<int> dist;  
};
```

Klasės panaudojimo pavyzdys

```
#include <iostream>
#include <random>
#include <vector>
#include "RandInt.hpp" // RandInt klasės deklaracija

int main() {
    constexpr int max = 9;
    RandInt rnd {0, max}; // sukuriame atsitiktinių skaičių generatorių

    std::vector<int> histogram(max + 1); // sukurti reikiamo dydžio vektorių
    for (int i=0; i!=100; ++i)
        ++histogram[rnd()]; // užpildome histogramą dažniais skaičių iš [0: max]

    for (auto i = 0; i!=histogram.size(); ++i) { // `nubrėžiame` stulpelinę diagramą
        std::cout << i << '\t';
        for (int j = 0; j!=histogram[i]; ++j)
            std::cout << '*';
        std::cout << std::endl;
    }
}
```

Klausimai ?

