C++17 i STL

Liczby pseudolosowe

Liczby pseudolosowe

- Standardowe narzędzia do generowania liczb pseudolosowych znajdują się w pliku nagłówkowym **<random>**.
- Liczby losowe są sekwencją wartości wytworzonych przez generator pseudolosowy przy użyciu matematycznych wzorów.
- Zastosowanie liczb losowych:
  - symulacje,
  - gry,
  - algorytmach probablistyczne,
  - kryptografia,
  - testowanie.

Mechanizmy losowości

- **Równomierny generator liczb losowych** to obiekt funkcyjny zwracający wartości całkowite bez znaku z rozkładem jednostajnym dla zadanego zakresu.
- **Mechanizm liczb losowych** jest równomiernym generatorem liczb, który można utworzyć w domyślnym stanie E{} lub w stanie określonym przez ziarno (ang. seed) E{s}.
- **Adaptacja mechanizmu liczb losowych** to mechanizm generowania liczb losowych, który pobiera wartości wytwarzane przez inny mechanizm generowania liczb losowych i przepuszcza je przez własny algorytm w celu utworzenia zbioru wartości o innych cechach losowości.
- **Rozkład liczb losowych** to obiekt funkcyjny zwracający wartości o rozkładzie zgodnym z powiązaną matematyczną funkcją gęstości prawdopodobieństwa.

Mechanizmy losowości

- Dla programisty generator liczb losowych to mechanizm połączony z rozkładem – mechanizm wytwarza równomiernie rozłożoną sekwencję wartości, a rozkład modyfikuje jej kształt do pożądanej postaci.
- Przykład (rozkład normalny `normal_distribution` z domyślnym mechanizmem `default_random_engine`):

```
auto gen = bind(
    normal_distribution<double>{15,4.0},
    default_random_engine{}
);
…
for (int i=0; i<500; ++i) cout << gen();
```

Mechanizmy losowości

- Najczęściej programiści potrzebują prostego równomiernego rozkładu liczb całkowitych z określonego przedziału:

```
// utwórz domyślny mechanizm losowości
std::default_random_engine dre;
// użyj mechanizmu do wygenerowania liczb
// całkowitych ze zbioru {10, …, 20}
std::uniform_int_distribution<int>
    di(10,20);
…
std::cout << di(dre) << std::endl;
```

Mechanizmy losowości

- Często też programiści potrzebują prostego równomiernego rozkładu liczb zmiennoprzecinkowych z określonego przedziału:

```
// utwórz domyślny mechanizm losowości
std::default_random_engine dre;
// użyj mechanizmu do wygenerowania liczb
// całkowitych z przedziału [-1, 1)
std::uniform_real_distribution<double>
    dr(-1, 1);
…
std::cout << dr(dre) << std::endl;
```

Mechanizmy i rozkłady

- **Mechanizmy** służą jako stanowe źródło losowości. Są to obiekty funkcyjne, zdolne do generowania dodatnich wartości z jednorodnym rozkładem wzdłuż określonego przedziału.
- **Rozkłady** służą do określania gęstości wartości losowych w zakresie określonym parametrami podanymi przez użytkownika.
- Liczbę losową generuje się za pośrednictwem wywołania operatora wywołania funkcji `operator()` na rzecz obiektu dystrybucji, z mechanizmem losowości jako argumentem wywołania.
- Jeśli nie chcemy zaczynać od przewidywalnej wartości liczbowej, powinniśmy jawnie ustawić przypadkowy stan generatora (na podstawie jakiegoś czynnika niezależnego od samego kodu programu, na przykład liczby milisekund pomiędzy dwoma kliknięciami przycisku myszy) - do konstruktora mechanizmu losowości trzeba przekazać **zarodek losowości**:

```
unsigned int seed = ... ;
std::default_random_engine dre(seed);
```

Ostrożnie z tymczasowymi obiektami mechanizmu losowości

- Nie powinno się przekazywać do algorytmów obiektu mechanizmu losowości tworzonego wyłącznie na potrzeby wywołania algorytmu (czyli na przykład obiektu tymczasowego). – za każdym razem przy tworzeniu i inicjalizowaniu mechanizmu losowości przybiera on taki sam w pełni określony stan.

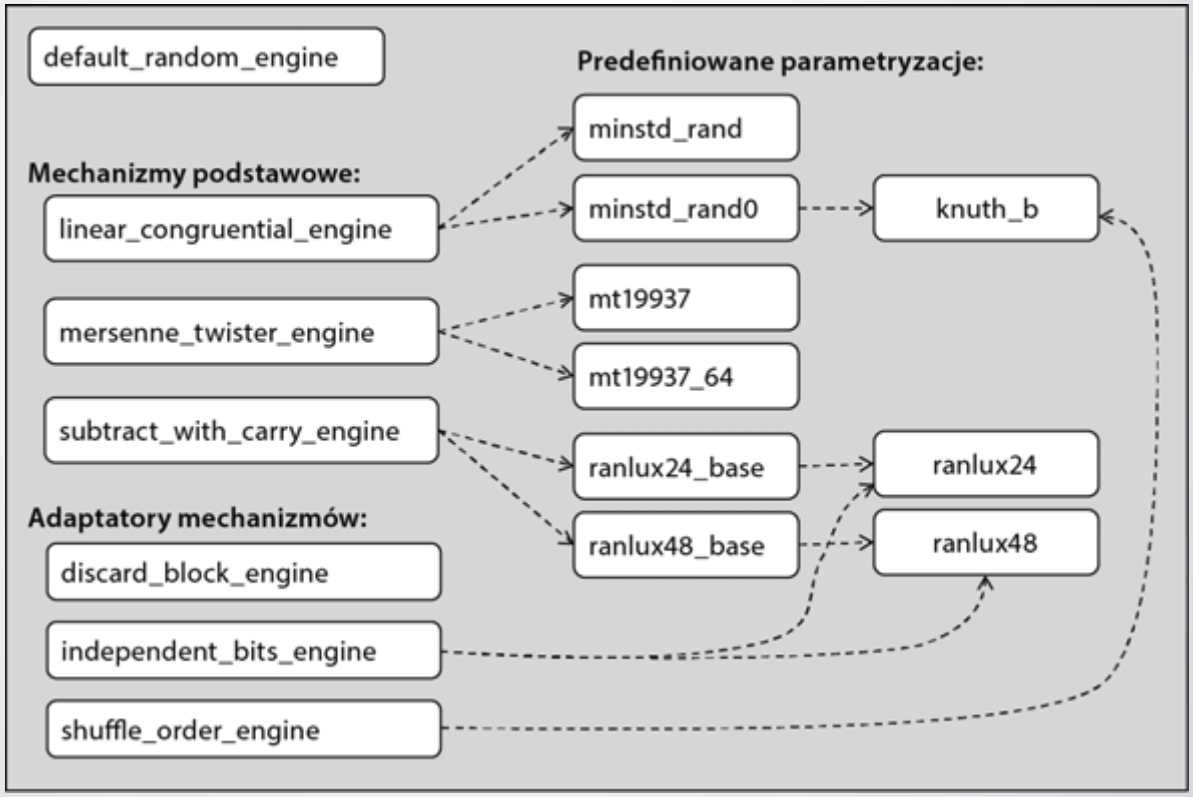
Nie używaj mechanizmów losowości bez rozkładów

- Naiwny programista zapytałby teraz: po co nam rozkłady, czy nie wystarczy używać bezpośrednio mechanizmów losowości i generowanych przez nich wartości losowych? A jeśli zakres generowanych wartości jest nieodpowiedni, czy nie wystarczy po prostu ograniczyć go operacją modulo (operator %) na wartościach wynikowych?
- Technika wyznaczania liczb losowych przez wyrażenie rand()%n w praktyce zawodzi z dwóch powodów:
  - wiele implementacji generatorów pseudolosowych daje reszty z małych n niebędące mocno losowe (na przykład nie jest rzadkością, że kolejne wyniki rand() są na przemian parzyste i nieparzyste);
  - jeśli wartość n jest duża, a wartość maksymalna generowanych wartości nie jest podzielna bez reszty przez n, niektóre reszty z dzielenia będą pojawiać się częściej niż inne.

Mechanizmy losowości

- Biblioteka standardowa C++ udostępnia 16 mechanizmów wartości losowych, które w połączeniu z dystrybucjami nadają się do generowania liczb losowych albo tasowania sekwencji wartości.
- Mechanizm losowości jest stanowy – jego stan definiuje sekwencję wartości losowych (które nie są liczbami losowymi); każde wywołanie funkcji (za pośrednictwem operatora wywołania funkcji operator()) zwraca kolejną wartość losową z sekwencji i zmienia wewnętrzny stan obiektu mechanizmu losowości.
- Przejścia pomiędzy stanami i generowane wartości są ściśle określone – jedynym wyjątkiem jest tu mechanizm domyślny, który może być wybrany różnie na różnych platformach.

Mechanizmy losowości



Mechanizmy losowości

- Mechanizm losowości jest obiektem funkcyjnym, to znaczy obiektem zachowującym się jak funkcja.
- Przykład:

```
void printNumbers (default_random_engine& dre)
{
    for (int i=0; i<6; ++i)
        cout << dre() << " „;
    cout << endl;
}
```

Mechanizmy losowości w szczegółach

- **Mechanizmy podstawowe** implementujące różne algorytmy generowania wartości losowych:
  - **linear_congruential_engine**,
  - **mersenne_twister_engine**,
  - **subtract_with_carry_engine**.
- **Adaptatory mechanizmów** stosowane do mechanizmów podstawowych:
  - **discard_block_engine** adaptująca mechanizm przez każdorazowe odrzucanie zadanej liczby wygenerowanych wartości,
  - **independent_bits_engine** adaptująca mechanizm pod kątem generowania wartości losowych o określonej liczbie bitów,
  - **shuffle_order_engine** adaptująca mechanizm przez permutację kolejności wartości generowanych przez mechanizm.
- **Adaptatory z predefiniowanymi parametrami**.

Rozkłady

- Rozkłady przekształcają wartości losowe generowane przez mechanizmy losowości na prawdziwie użyteczne liczby losowe.
- Dystrybucja prawdopodobieństwa generowanych liczb zależy od typu rozkładu, dodatkowo parametryzowanego zgodnie z potrzebami programisty.
- Zdefiniowane rozkłady:
  - rozkład jednostajny (uniform_int_distribution, uniform_real_distribution);
  - rozkład Bernoulliego (bernoulli_distribution, binomial_distribution, geometric_distribution);
  - rozkład Poissona (poisson_distribution, exponential_distribution);
  - rozkład normalny (normal_distribution, chi_squared_distribution, student_t_distribution);
  - rozkład próbkujący (discrete_distribution).

Rozkłady

- Przykłady:

```
uniform_int_distribution<> d(0, 20);
d.a();
d.b();

std::knuth_b e;

std::uniform_real_distribution<> ud(0, 10);

std::normal_distribution<> nd;
```

Urządzenie losowe

- Jeśli implementacja ma możliwość dostarczenia prawdziwie losowego generatora liczb, to źródło tych liczb ma postać równomiernego generatora liczb losowych o nazwie random_device.

Losowanie liczb w stylu C

- W pliku nagłówkowym <cstdlib> znajduje się prosta podstawa do generowania liczb losowych:
  - funkcja rand() – liczba pseudolosowa z przedziału od 0 do RAND_MAX;
  - funkcja srand(unsigned) – podanie ziarna dla generatora liczb pseudolosowych.
- Utworzenie dobrego generatora liczb losowych nie jest łatwe i niestety nie wszystkie systemy zawierają dobrą funkcję rand().
- Często wyniki do przyjęcia daje operacja:
  `int((rand() / (RAND_MAX + 1.0)) * n)`