Random в языке C++

При изучении языка C++ многие испытывают трудности при использовании функций и классов из библиотеки <***random***> и часто делают это неправильно, либо вообще используют функции по типу ***rand()***. Ниже будут рассмотрены основные принципы работы генераторов случайных чисел и разобраны типичные ошибки при их применении в программировании на языке C++.

1.1 Немного о генерации случайных чисел

Генерация истинно случайных чисел — сложный и вычислительно дорогой процесс. Для этого обычно собирается разная информация с процессора и прочего оборудования. С другой стороны, во многих приложениях не нужно иметь настоящие случайные числа, достаточно иметь числа, которые выглядят как случайные (такие числа будем называть ***псевдослучайными***). Именно генерация таких чисел и реализована в подавляющем большинстве библиотек в разных языках.

Простейший способ генерировать такие числа — взять последовательность x_n, определенную по правилу

$$x_n = (ax_{n-1} + b) \mod c,$$

где a, b, c — некоторые коэффициенты. При их правильном подборе получающая последовательность чисел статистически будет выглядеть как случайная. Однако понятно, что вся последовательность детерминированная и целиком задается x_0. Данное число обычно называют «сидом» (***seed***). Описанный генератор сейчас используется редко (один из его недостатков — его период не более c), но важно понимать, что все генераторы псевдослучайных чисел обладают тем же свойством — **вся последовательность чисел задается сидом**. Если при каждом запуске программы используется один и тот же сид, то и случайные числа в программе будут одними и теми же.

1.2 Функция rand из языка C (си)

Функция ***rand*** из языка C (си) при каждом вызове генерирует очередное x_n(не обязательно по правилу, описанному выше, может быть и другое) и возвращает его. Функция ***srand*** позволяет установить значение x_0. При попытках гуглить вы могли встретить что-то похожее на ***srand(time(0))***.

Здесь в качестве сида будет установлено текущее время, поэтому при каждом запуске программы числа будут разными. Сейчас разберёмся, почему же так не стоит делать.

Функцию ***rand()*** не стоит использовать по разным причинам. Во-первых, стандарт не фиксирует её реализацию, поэтому на разных платформах генерируемые числа будут разными для одной и той же программы с одним и тем же сидом (например, на *windows* и *linux*). Более того, на *windows* обычно максимальное значение ***rand*** ограничено величиной 32768, что очень мало, и вы, наверное, могли встретить такой «костыль»:

```
int random_number = rand() | (rand() << 15);
```

Листинг 1.1: «Костыль» для генерации большого случайного числа

Данный «костыль» на листинге 1.1 возможно и сгенерирует большое число, но он неправильно сработает на *linux*, т.к. там максимальное значение ***rand*** ограничено только диапазоном типа **int**. Вдобавок, последовательность, генерируемая ***rand***, обычно достаточно примитивна, в частности она может иметь довольно малый период.

По той же причине не рекомендую использовать **std::random_shuffle**, т.к. она реализована через ***rand***. Вместо неё есть **std::shuffle**.

Более подробно про ***rand*** можно посмотреть здесь.

1.3 Генераторы и распределения в языке C++

Первое, что нужно помнить — в C++ есть *генераторы* и *распределения*. Предположим, что вы хотите сгенерировать целое равномерно распределенное на отрезке $[\mathbf{1}, \mathbf{n}]$ число. В языке *C* (си) для этого нужно делать что-то такое:

```
int random_number = rand() % n + 1;
```

Листинг 1.2: Генерация числа на $[\mathbf{1}, \mathbf{n}]$

Представленный вариант на листинге 1.2 не совсем корректен (поскольку не всегда диапазон ***rand*** будет кратен n), а также плохо обобщается на другие распределения(допустим вы захотите равномерное распределение заменить на нормальное).

Генератор ведёт себя примерно как ***rand*** — возвращает целое число в каком-то диапазоне, например (см. листинг 1.3 на след. странице):

```
std::default_random_engine gen;
int x = gen();
```

Листинг 1.3: Генератор в языке C++

В стандартной библиотеке есть много разных генераторов. Я рекомендую использовать ***std::mt19937***. Этот генератор реализует вихрь Мерсенна, он генерирует последовательность «хорошего» качества и достаточно быстр. В качестве параметра ему можно передать ранее упомянутый сид (может отличаться от использованного ниже):

```
std::mt19937 gen(45218965);
```

Листинг 1.4: Вихрь Мерсенна с указанным сидом

Для генерации числа с нужным распределением следует создать соответствующий объект, например:

```
std::mt19937 gen(45218965);
// для генерации равномерно распределённого
// целого числа на [1, n]
std::uniform_int_distribution<int> dist(1, n);
int x = dist(gen);
```

Листинг 1.5: Генерация целого числа

Аналогичным способом можно сгенерировать равномерно распределённое на отрезке вещественное число (см. листинг 1.6):

```
std::mt19937 gen(45218965);
// для генерации равномерно распределённого
// вещественного числа
// на отрезке [1.2, 10.5]
std::uniform_real_distribution<double> dist(1.2, 10.5);
double x = dist(gen);
```

Листинг 1.6: Генерация вещественного числа

1.4 Распространённые ошибки

1.4.1 Каждый раз создаётся новый генератор

Допустим у вас есть рандомизированный алгоритм, который много раз вызывает функцию **f**, в которой нужно генерировать случайные числа. **Неправильно** будет сделать следующее:

```
int f() {
    std::mt19937 gen(some_seed);
    // use gen
}
```

Листинг 1.7: Неправильное использование генератора

Как мы уже знаем, в таком случае во всех вызовах **f** будет использоваться одна и та же последовательность чисел. **Не пытайтесь** решать эту проблему созданием нескольких генераторов с разными сидами или как-то передавать его извне, вместо этого **используйте один генератор** во всём алгоритме и передавайте его в функцию **по ссылке** или же **по указателю** (как это написано в примере ниже):

```
int f(std::mt19937* gen) {
    // use gen
}
```

Листинг 1.8: Правильное использование генератора

1.4.2 Копирование генератора

Данная ошибка может возникнуть, если код, написанный на листинге 1.8, написать следующим образом:

```
int f(std::mt19937 gen) {
    // use gen
}
```

Листинг 1.9: Копирование генератора

В этом случае мы вернёмся к предыдущей проблеме. Вся та же последовательность чисел определяется уже сгенерированной. При копировании генератора копируется его внутреннее состояние (например, если это будет простейший генератор из начала, то будет копироваться последнее сгенерированное значение x_n). Это будет означать что в приведённом ниже коде

```
f(gen);
f(gen);
f(gen);
//use gen;
```

Листинг 1.10: Копирование генератора

во всех трёх случаях и после них будет использоваться одна и та же последовательность чисел. Поэтому правильнее будет передавать генератор по ссылке или указателю, тем самым его состояние будет меняться между вызовами.

1.4.3 Сид не указывается / устанавливается случайным образом

В подавляющем большинстве случаев ваша программа должна быть детерминированной: вести себя одинаково на одних и тех же входных данных при каждом запуске. Такую программу будет легче оттестировать и использовать в дальнейшем. Этого может не быть, если написать следующий код:

```
std::random_device rd;
std::mt19937 gen(rd());
```

Листинг 1.11: Пример недетерминированности

random_device обычно генерирует «настоящие» случайные числа (поэтому и работает медленно). Он может понадобиться в какой-нибудь криптографии, например, в нашем случае он не нужен.