

Ravesli [Ravesli](#)


- [Уроки по C++](#)
- [OpenGL](#)
- [SFML](#)
- [Qt5](#)
- [RegExr](#)
- [Ассемблер](#)
- [Купить .PDF](#)


## Урок №71. Генерация случайных чисел

 [Юрий](#) |

- [Уроки C++](#)

|

 Обновл. 15 Сен 2020 |

 109463

[↑](#)  30

Возможность генерировать случайные числа очень полезна в некоторых видах программ, в частности, в играх, программах научного или статистического моделирования. Возьмем, к примеру, игры без **рандомных** (или «случайных») **событий** — монстры всегда будут атаковать вас одинаково, вы всегда будете находить одни и те же предметы/артефакты, макеты темниц и подземелий никогда не будут меняться и т.д. В общем, сюжет такой игры не очень интересен и вряд ли вы будете в нее долго играть.

Оглавление:

1. [Генератор псевдослучайных чисел](#)
2. [Функции `srand\(\)` и `rand\(\)`](#)
3. [Стартовое число и последовательности в ГПСЧ](#)
4. [Генерация случайных чисел в заданном диапазоне](#)
5. [Какой ГПСЧ является хорошим?](#)
6. [Почему `rand\(\)` является посредственным ГПСЧ?](#)
7. [Отладка программ, использующих случайные числа](#)
8. [Рандомные числа в C++11](#)

## Генератор псевдослучайных чисел

Так как же генерировать случайные числа? В реальной жизни мы часто бросаем монетку (орел/решка), кости или перетасовываем карты. Эти события включают в себя так много физических переменных (например, сила тяжести, трение, сопротивление воздуха и т.д.), что они становятся почти невозможными для прогнозирования/контроля и выдают результаты, которые во всех смыслах являются случайными.

Однако компьютеры не предназначены для использования физических переменных — они не могут подбросить монетку, бросить кости или перетасовать реальные карты. Компьютеры живут в контролируемом электрическом мире, где есть только два значения (**true и false**), чего-то среднего между ними нет. По своей природе компьютеры предназначены для получения прогнозируемых результатов. Когда мы говорим компьютеру посчитать, сколько будет  $2 + 2$ , мы *всегда* хотим, чтобы ответом было 4 (не 3 и не 5).

Следовательно, компьютеры неспособны генерировать случайные числа. Вместо этого они могут имитировать случайность, что достигается с помощью генераторов псевдослучайных чисел.

**Генератор псевдослучайных чисел** (сокр. «ГПСЧ») — это программа, которая принимает стартовое/начальное значение и выполняет с ним определенные математические операции, чтобы конвертировать его в другое число, которое совсем не связано со стартовым. Затем программа использует новое сгенерированное значение и выполняет с ним те же математические операции, что и с начальным числом, чтобы конвертировать его в еще одно новое число — третье, которое не связано ни с первым, ни со вторым. Применяя этот алгоритм к последнему сгенерированному значению, программа может генерировать целый ряд новых чисел, которые будут казаться случайными (при условии, что алгоритм будет достаточно сложным).

На самом деле, написать простой ГПСЧ не так уж и сложно. Вот небольшая программа, которая генерирует 100 рандомных чисел:

```
1  #include <iostream>
2
3  unsigned int PRNG()
4  {
5      // Наше стартовое число - 4 541
6      static unsigned int seed = 4541;
7
8      // Берем стартовое число и, с его помощью, генерируем новое значение.
9      // Из-за использования очень больших чисел (и переполнения) угадать следующее число
10     seed = (8253729 * seed + 2396403);
11
12     // Берем стартовое число и возвращаем значение в диапазоне от 0 до 32767
13     return seed % 32768;
14 }
15
16 int main()
17 {
18     // Выводим 100 случайных чисел
19     for (int count=0; count < 100; ++count)
20     {
21         std::cout << PRNG() << "\t";
22
23         // Если вывели 5 чисел, то вставляем символ новой строки
24         if ((count+1) % 5 == 0)
25             std::cout << "\n";
26     }
27 }
```

Результат выполнения программы:

18256	4675	32406	6217	27484
975	28066	13525	25960	2907
12974	26465	13684	10471	19898
12269	23424	23667	16070	3705
22412	9727	1490	773	10648
1419	8926	3473	20900	31511
5610	11805	20400	1699	24310
25769	9148	10287	32258	12597
19912	24507	29454	5057	19924
11591	15898	3149	9184	4307
24358	6873	20460	2655	22066
16229	20984	6635	9022	31217
10756	16247	17994	19069	22544
31491	16214	12553	23580	19599
3682	11669	13864	13339	13166
16417	26164	12711	11898	26797
27712	17715	32646	10041	18508
28351	9874	31685	31320	11851
9118	26193	612	983	30378
26333	24688	28515	8118	32105

Каждое число кажется случайным по отношению к предыдущему. Главный недостаток этого алгоритма — его примитивность.

## Функции `srand()` и `rand()`

Языки Си и C++ имеют свои собственные встроенные генераторы случайных чисел. Они реализованы в 2-х отдельных функциях, которые находятся в заголовочном файле `cstdlib`:

- ➔ Функция `srand()` устанавливает передаваемое пользователем значение в качестве стартового. `srand()` следует вызывать только один раз — в начале программы (обычно в верхней части функции `main()`).
- ➔ Функция `rand()` генерирует следующее случайное число в последовательности. Оно будет находиться в диапазоне от 0 до `RAND_MAX` (константа в `cstdlib`, значением которой является 32767).

Вот пример программы, в которой используются обе эти функции:

```
1 #include <iostream>
2 #include <cstdlib> // для функций rand() и srand()
3
4 int main()
5 {
6     srand(4541); // устанавливаем стартовое значение - 4 541
7
8 }
```

```

9 // Выводим 100 случайных чисел
10 for (int count=0; count < 100; ++count)
11 {
12     std::cout << rand() << "\t";
13
14     // Если вывели 5 чисел, то вставляем символ новой строки
15     if ((count+1) % 5 == 0)
16         std::cout << "\n";
17 }

```

Результат выполнения программы:

14867	24680	8872	25432	21865
17285	18997	10570	16397	30572
22339	31508	1553	124	779
6687	23563	5754	25989	16527
19808	10702	13777	28696	8131
18671	27093	8979	4088	31260
31016	5073	19422	23885	18222
3631	19884	10857	30853	32618
31867	24505	14240	14389	13829
13469	11442	5385	9644	9341
11470	189	3262	9731	25676
1366	24567	25223	110	24352
24135	459	7236	17918	1238
24041	29900	24830	1094	13193
10334	6192	6968	8791	1351
14521	31249	4533	11189	7971
5118	19884	1747	23543	309
28713	24884	1678	22142	27238
6261	12836	5618	17062	13342
14638	7427	23077	25546	21229

## Стартовое число и последовательности в ГПСЧ

Если вы запустите вышеприведенную программу (генерация случайных чисел) несколько раз, то заметите, что в результатах всегда находятся одни и те же числа! Это означает, что, хотя каждое число в последовательности кажется случайным относительно предыдущего, вся последовательность не является случайной вообще! А это, в свою очередь, означает, что наша программа полностью предсказуема (одни и те же значения ввода приводят к одним и тем же значениям вывода). Бывают случаи, когда это может быть полезно или даже желательно (например, если вы хотите, чтобы научная симуляция повторялась, или вы пытаетесь исправить причины сбоя вашего генератора случайных подземелий в игре).

Но в большинстве случаев это не совсем то, что нам нужно. Если вы пишете игру типа *Hi-Lo* (где у пользователя есть 10 попыток угадать число, а компьютер говорит ему, насколько его предположения близки или далеки от реального числа), вы бы не хотели, чтобы программа выбирала одни и те же числа каждый раз. Поэтому давайте более подробно рассмотрим, почему это происходит и как это можно исправить.

Помните, что каждое новое число в последовательности ГПСЧ генерируется исходя из предыдущего определенным способом. Таким образом, при постоянном начальном числе ГПСЧ всегда будет генерировать одну и ту же последовательность! В программе, приведенной выше, последовательность чисел всегда одинакова, так как стартовое число всегда равно 4541.

Чтобы это исправить нам нужен способ выбрать стартовое число, которое не будет фиксированным значением. Первое, что приходит на ум — использовать случайное число! Это хорошая мысль, но если нам нужно случайное число для генерации случайных чисел, то это какой-то замкнутый круг, вам не кажется? Оказывается, нам не обязательно использовать случайное стартовое число — нам просто нужно выбрать что-то, что будет меняться каждый раз при новом запуске программы. Затем мы сможем использовать наш ГПСЧ для генерации уникальной последовательности случайных чисел исходя из уникального стартового числа.

Общепринятым решением является использование системных часов. Каждый раз, при запуске программы, время будет другое. Если мы будем использовать значение времени в качестве стартового числа, то наша программа всегда будет генерировать разную последовательность чисел при каждом новом запуске!

В языке Си есть **функция `time()`**, которая возвращает в качестве времени общее количество секунд, прошедшее от полуночи 1 января 1970 года. Чтобы использовать эту функцию, нам просто нужно подключить заголовочный файл `ctime`, а затем инициализировать функцию `srand()` вызовом функции `time(0)`.

Вот вышеприведенная программа, но уже с использованием функции `time(0)` в качестве стартового числа:

```
1  #include <iostream>
2  #include <cstdlib> // для функций rand() и srand()
3  #include <ctime> // для функции time()
4
5  int main()
6  {
7      srand(static_cast<unsigned int>(time(0))); // устанавливаем значение системных часов
8
9      for (int count=0; count < 100; ++count)
10     {
11         std::cout << rand() << "\t";
12
13         // Если вывели 5 чисел, то вставляем символ новой строки
14         if ((count+1) % 5 == 0)
15             std::cout << "\n";
16     }
17 }
```

Теперь наша программа будет генерировать разные последовательности случайных чисел! Попробуйте сами.

## Генерация случайных чисел в заданном диапазоне

В большинстве случаев нам не нужны случайные числа между 0 и `RAND_MAX` — нам нужны числа между двумя другими значениями: `min` и `max`. Например, если нам нужно симитировать бросок кубика, то диапазон значений будет невелик: от 1 до 6.

Вот небольшая функция, которая конвертирует результат функции `rand()` в нужный нам диапазон значений:

```
1 // Генерируем случайное число между значениями min и max.
2 // Предполагается, что функцию srand() уже вызывали
3 int getRandomNumber(int min, int max)
4 {
5     static const double fraction = 1.0 / (static_cast<double>(RAND_MAX) + 1.0);
6     // Равномерно распределяем случайное число в нашем диапазоне
7     return static_cast<int>(rand() * fraction * (max - min + 1) + min);
8 }
```

Чтобы симитировать бросок кубика, вызываем функцию `getRandomNumber(1, 6)`.

## Какой ГПСЧ является хорошим?

Как мы уже говорили, генератор случайных чисел, который мы написали выше, не является очень хорошим. Сейчас рассмотрим почему.

**Хороший ГПСЧ должен иметь ряд свойств:**

**Свойство №1:** ГПСЧ должен генерировать каждое новое число с примерно одинаковой вероятностью. Это называется **равномерностью распределения**. Если некоторые числа генерируются чаще, чем другие, то результат программы, использующей ГПСЧ, будет предсказуем! Например, предположим, вы пытаетесь написать генератор случайных предметов для игры. Вы выбираете случайное число от 1 до 10, и, если результатом будет 10, игрок получит крутой предмет вместо среднего. Шансы должны быть 1 к 10. Но, если ваш ГПСЧ неравномерно генерирует числа, например, десятки генерируются чаще, чем должны, то ваши игроки будут получать более редкие предметы чаще, чем предполагалось, и сложность, и интерес к такой игре автоматически уменьшаются.

Создать ГПСЧ, который бы генерировал равномерные результаты — сложно, и это одна из главных причин, по которым ГПСЧ, который мы написали в начале этого урока, не является очень хорошим.

**Свойство №2:** Метод, с помощью которого генерируется следующее число в последовательности, не должен быть очевиден или предсказуем. Например, рассмотрим следующий алгоритм ГПСЧ: `num = num + 1`. У него есть равномерность распределения случайных чисел, но это не спасает его от примитивности и предсказуемости!

**Свойство №3:** ГПСЧ должен иметь хорошее диапазонное распределение чисел. Это означает, что маленькие, средние и большие числа должны возвращаться случайным образом. ГПСЧ, который возвращает все маленькие числа, а затем все большие — предсказуем и приведет к предсказуемым результатам.

**Свойство №4:** Период циклического повторения значений ГПСЧ должен быть максимально большим. Все ГПСЧ являются циклическими, т.е. в какой-то момент последовательность генерируемых чисел начнет повторяться. Как упоминалось ранее, ГПСЧ являются детерминированными, и с одним

значением ввода мы получим одно и то же значение вывода. Подумайте, что произойдет, когда ГПСЧ сгенерирует число, которое уже ранее было сгенерировано. С этого момента начнется дублирование последовательности чисел между первым и последующим появлением этого числа. Длина этой последовательности называется **периодом**.

Например, вот представлены первые 100 чисел, сгенерированные ГПСЧ с плохой периодичностью:

112	9	130	97	64
31	152	119	86	53
20	141	108	75	42
9	130	97	64	31
152	119	86	53	20
141	108	75	42	9
130	97	64	31	152
119	86	53	20	141
108	75	42	9	130
97	64	31	152	119
86	53	20	141	108
75	42	9	130	97
64	31	152	119	86
53	20	141	108	75
42	9	130	97	64
31	152	119	86	53
20	141	108	75	42
9	130	97	64	31
152	119	86	53	20
141	108	75	42	9

Заметили, что он сгенерировал 9, как второе число, а затем как шестнадцатое? ГПСЧ застревает, генерируя последовательность между этими двумя 9-ми: 9-130-97-64-31-152-119-86-53-20-141-108-75-42- (повтор).

Хороший ГПСЧ должен иметь длинный период для *всех* стартовых чисел. Разработка алгоритма, соответствующего этому требованию, может быть чрезвычайно сложной — большинство ГПСЧ имеют длинные периоды для одних начальных чисел и короткие для других. Если пользователь выбрал начальное число, которое имеет короткий период, то и результаты будут соответствующие.

Несмотря на сложность разработки алгоритмов, отвечающих всем этим критериям, в этой области было проведено большое количество исследований, так как разные ГПСЧ активно используются в важных отраслях науки.

## Почему rand() является посредственным ГПСЧ?

Алгоритм, используемый для реализации rand(), может варьироваться в разных компиляторах, и, соответственно, результаты также могут быть разными. В большинстве реализаций rand() используется [Линейный Конгруэнтный Метод](#) (сокр. «ЛКМ»). Если вы посмотрите на первый пример в этом уроке,

то заметите, что там, на самом деле, используется ЛКМ, хоть и с намеренно подобранными плохими константами.

Одним из основных недостатков функции `rand()` является то, что `RAND_MAX` обычно устанавливается как 32767 (15-битное значение). Это означает, что если вы захотите сгенерировать числа в более широком диапазоне (например, 32-битные целые числа), то функция `rand()` не подойдет. Кроме того, она не подойдет, если вы захотите сгенерировать случайные числа [типа с плавающей запятой](#) (например, между 0.0 и 1.0), что часто используется в статистическом моделировании. Наконец, функция `rand()` имеет относительно короткий период по сравнению с другими алгоритмами.

Тем не менее, этот алгоритм отлично подходит для изучения программирования и для программ, в которых высококлассный ГПСЧ не является необходимостью.

Для приложений, где требуется высококлассный ГПСЧ, рекомендуется использовать [алгоритм Вихрь Мерсенна](#) (англ. «*Mersenne Twister*»), который генерирует отличные результаты и относительно прост в использовании.

## Отладка программ, использующих случайные числа

Программы, которые используют случайные числа, трудно [отлаживать](#), так как при каждом запуске такой программы мы будем получать разные результаты. А чтобы успешно проводить отладку программ, нужно удостовериться, что наша программа выполняется одинаково при каждом её запуске. Таким образом, мы сможем быстро узнать расположение ошибки и изолировать этот участок кода.

Поэтому, проводя отладку программы, полезно использовать в качестве стартового числа (с использованием функции `srand()`) определенное значение (например, 0), которое вызовет ошибочное поведение программы. Это будет гарантией того, что наша программа каждый раз генерирует одни и те же результаты (что значительно облегчит процесс отладки). После того, как мы найдем и исправим ошибку, мы сможем снова использовать системные часы для генерации рандомных результатов.

## Рандомные числа в C++11

В C++11 добавили тонну нового функционала для генерации случайных чисел, включая алгоритм Вихрь Мерсенна, а также разные виды генераторов случайных чисел (например, равномерные, генератор Poisson и пр.). Доступ к ним осуществляется через подключение заголовочного файла `random`. Вот пример генерации случайных чисел в C++11 с использованием Вихря Мерсенна:

```
1 #include <iostream>
2 // #include <ctime> // раскомментируйте, если используете Code::Blocks
3 #include <random> // для std::random_device и std::mt19937
4
5 int main()
6 {
7     std::random_device rd;
8     std::mt19937 mersenne(rd()); // инициализируем Вихрь Мерсенна случайным стартовым
9
10 // Примечание: Из-за одного бага в компиляторе Code::Blocks (если вы используете Code::
```



```
11 // std::mt19937 mersenne(static_cast<unsigned int>(time(0))); // инициализируем Вихрь
12
13 // Выводим несколько случайных чисел
14 for (int count = 0; count < 48; ++count)
15 {
16     std::cout << mersenne() << "\t";
17
18     // Если вывели 5 чисел, то вставляем символ новой строки
19     if ((count + 1) % 5 == 0)
20         std::cout << "\n";
21 }
22 }
```

Вихрь Мерсенна генерирует случайные 32-битные целые числа unsigned (а не 15-битные целые числа, как в случае с `rand()`), что позволяет использовать гораздо больший диапазон значений. Существует также версия (`std::mt19937_64`) для генерации 64-битных целых чисел `unsigned`!

**Примечание для пользователей Visual Studio:** Реализация функции `rand()` в Visual Studio имеет один существенный недостаток — первое генерируемое случайное число не сильно отличается от стартового. Это означает, что, при использовании `time()` для генерации начального числа, первое число не будет сильно отличаться/изменяться от стартового и при последующих запусках. Есть простое решение: вызовите функцию `rand()` один раз и сбросьте результат. Затем вы сможете использовать `rand()`, как обычно в вашей программе.

Оценить статью:

★★★★★ (275 оценок, среднее: 4,83 из 5)

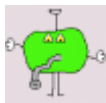


[← Урок №70. Операторы `break` и `continue`](#)

[Урок №72. Обработка некорректного пользовательского ввода](#) →



## Комментариев: 30



1. Дарья:

[22 октября 2020 в 17:20](#)

Подскажите, пожалуйста, а как можно сгенерировать случайные числа так, чтобы среди них были и отрицательные в том числе?

[Ответить](#)



1. *Геннадий:*

[3 ноября 2020 в 06:56](#)

использовать `uniform_int_distribution` <> "любое имя"("нужный диапазон")

[Ответить](#)



2. *Влад:*

[22 мая 2020 в 14:24](#)

Подскажите обязательно ли подключать библиотеку `cstdlib` для `rand`.

В Visual Studio и CodeBlocks достаточно подключить `iostream` чтобы `rand` заработал.

Или в `iostream` уже подюлючена `cstdlib`?

[Ответить](#)



1. *Ruslan:*

[24 сентября 2020 в 12:36](#)

Присоединяюсь к вопросу, аналогично:

с функцией `time ()` нужно ли подключать `<ctime>`,

с функцией `exit ()` нужно ли подключать `<cstdlib>`,

с функцией `sqrt ()` нужно ли подключать `<cmath>`,

т.к. компилируется без предупреждений, работают и без этих заголовочных файлов? (впрочем как и с ними).

Нужно ли указывать перед этими функциями `std::` ( работает и с указанием и без) ?

Компилятор `g++`.

[Ответить](#)



3. *nickatin:*

[13 мая 2020 в 21:16](#)

Очень крутой сайт!!! Автору огромное спасибо за труды. Дошел до 6 главы занятий, все понятно но не много не понял курс с генерацией случайных чисел, какие то сложные алгоритмы))

Вопрос вот написал маленький код для случайных чисел кубика от 1 до 6, в чем он плох, распределение будет ужасное? Прошу посмотреть и прокомментировать)

```
1 #include <iostream>
2
3 int main(){
4     using namespace std;
5     srand(time(0));
6     int x = rand();
7     int y = (x % 6) + 1;
8     std::cout << y <<std::endl;
```

```
9  
10     return 0;  
11 }
```

### [Ответить](#)



1. *Максим:*

[31 мая 2020 в 23:57](#)

Здесь уже задавали вопрос про "рандом от остатка", но так никто и не ответил.  
На мой (ученический) глаз рандом у чисел получается хороший.

Набросал программку для определения распределения случайных чисел от 0 до 10.  
Чем больше выставить количество итераций, тем равномернее получается распределение чисел.

```
1 // Анализ ГСПЧ по "остатку от деления" rand()  
2 // min = 1, max = 10  
3  
4 #include <iostream>  
5 #include <cstdlib>  
6 #include <ctime>  
7  
8 int main()  
9 {  
10     srand(static_cast<unsigned int>(time(0)));  
11  
12     int min = 1;  
13     int max = 10;  
14  
15     int value; // выпадаемое значение  
16  
17     // счетчики каждого отдельного числа  
18     int count_1 = 0;  
19     int count_2 = 0;  
20     int count_3 = 0;  
21     int count_4 = 0;  
22     int count_5 = 0;  
23     int count_6 = 0;  
24     int count_7 = 0;  
25     int count_8 = 0;  
26     int count_9 = 0;  
27     int count_10 = 0;  
28  
29     for(int count = 1; count <= 100; ++count)  
30     {  
31         value = rand()%(max-min+1)+min;  
32         std::cout << value << "\t";
```

```
33
34     if (value == 1) count_1++;
35     if (value == 2) count_2++;
36     if (value == 3) count_3++;
37     if (value == 4) count_4++;
38     if (value == 5) count_5++;
39     if (value == 6) count_6++;
40     if (value == 7) count_7++;
41     if (value == 8) count_8++;
42     if (value == 9) count_9++;
43     if (value == 10) count_10++;
44
45     if (count % 20 == 0)
46         std::cout << "\n";
47 }
48
49 int total = count_1 + count_2 + count_3 + count_4 + count_5 + count_6 +
50
51 // "проверка на дурака", должно быть равно количеству итераций
52 std::cout << "\nThe total number is " << total << "\n";
53
54 // выводим процент выпадения каждого числа
55 std::cout << "1: " << (static_cast<double>(count_1) / total) * 100 << "%\n";
56 std::cout << "2: " << (static_cast<double>(count_2) / total) * 100 << "%\n";
57 std::cout << "3: " << (static_cast<double>(count_3) / total) * 100 << "%\n";
58 std::cout << "4: " << (static_cast<double>(count_4) / total) * 100 << "%\n";
59 std::cout << "5: " << (static_cast<double>(count_5) / total) * 100 << "%\n";
60 std::cout << "6: " << (static_cast<double>(count_6) / total) * 100 << "%\n";
61 std::cout << "7: " << (static_cast<double>(count_7) / total) * 100 << "%\n";
62 std::cout << "8: " << (static_cast<double>(count_8) / total) * 100 << "%\n";
63 std::cout << "9: " << (static_cast<double>(count_9) / total) * 100 << "%\n";
64 std::cout << "10: " << (static_cast<double>(count_10) / total) * 100 << "%\n";
65
66 return 0;
67 }
```

#### [Ответить](#)



1. *Олег:*

[17 июля 2020 в 08:27](#)

Похоже, массивы еще не изучали...

#### [Ответить](#)



2. *RexStar:*

[16 августа 2020 в 02:24](#)

Главное здесь скорость... Операции деления и остатка "весят" на порядок выше других математических операций (не всех 😊)

[Ответить](#)



4. *Furxie Fluke:*

[6 мая 2020 в 04:38](#)

Придумал функцию конвертации randomного числа в диапазон значений чуть проще, без константной переменной и нескольких лишних вычислений

Пытался просто придумать сам как это реализовать математически, не смотря на пример в задании, и придумав — сравнил, был рад увидеть что итоги схожи ) Но ещё более рад был увидеть что придумал функцию попроще, с одинаковым функционалом :3

```
1 int randomBetween(double min, double max)
2 {
3     return rand() * ((max - min + 0.999) / RAND_MAX) + min;
4 }
```

\*0.999 — потому что если без неё то шестёрка будет выпадать только примерно раз в 32767 раз (в среднем), когда randomное значение достигнет своей границы. А это не то что нам нужно.

Не 1.0 — потому что тогда помимо шестёрки будет так же (хоть и редко) выпадать 7, тоже в среднем раз в 32767 раз, этого тоже стоит избегать.

Поэтому "идеальный" вариант 0.999, — потому что даже при максимальном значении randomного числа (32767), получится 6.999, что округлится к 6. И так же не будет потерян диапазон вероятности выпадения шестёрки, то есть шанс её выпадения не уменьшится, ну. разве что на 0.001 %, что вроде не столь значимо, и если добавить ещё девяток — будет ещё меньше

[Ответить](#)



5. *zashiki:*

[3 сентября 2019 в 19:49](#)

а чем для диапазона малых значений(от 1 до 6 и тд) хуже random от остатка, чем random с округлением дроби?

К примеру, вот так пытаюсь, выходит норм

```
1 srand(static_cast<unsigned int>(time(0)));
2 std::cout<<rand()%(max-min+1)+min;
```

[Ответить](#)



6. *Лев:*

[1 июня 2019 в 00:13](#)

Получилось добавить возможность указывать количество чисел, строк, min, max. Но главное, что это работает с Вихрем Мерсенна!)

Хотел спросить, не допустил ли каких-то потенциальных ошибок?

```
1 #include <iostream>
2 #include <random> // для std::random_device и std::mt19937
3
4 int main()
5 {
6     std::random_device rd;
7     std::mt19937_64 mersenne(rd()); // инициализируем Вихрь Мерсенна случайным ст.
8
9
10    int n, rows, min, max;
11    std::cout << "How many number: ";
12    std::cin >> n;
13
14    std::cout << "How many rows: ";
15    std::cin >> rows;
16
17    std::cout << "Enter min namber: ";
18    std::cin >> min;
19
20    std::cout << "Enter max namber: ";
21    std::cin >> max;
22
23    std::cout << "\n" << "Results: " << "\n" << "\n";
24
25
26    unsigned long long g{ 18446744073709551615 };
27    for (int count = 0; count < n; ++count)
28    {
29        static const long double fraction = 1.0 / (static_cast<long double>(g + 1));
30        std::cout << static_cast<unsigned long long>(mersenne() * fraction * (max
31
32        if ((count + 1) % rows == 0)
33            std::cout << "\n";
34    }
35
36
37    std::cin.clear();
38    std::cin.ignore(32767, '\n');
39    std::cin.get();
40 }
```

[Ответить](#)



7. [Алексей:](#)

[11 мая 2019 в 13:23](#)

При генерации через системные часы первое число всегда будет одинаковым? Или я не так что-то сделал?

```
1 #include <iostream>
2 #include <cstdlib>
3 #include <ctime>
4
5 int randDam(int min, int max)
6 {
7     static const double fraction = 1.0 / (static_cast<double>(RAND_MAX) + 1.0);
8     return static_cast<int>(rand() * fraction * (max - min + 1) + min);
9 }
10
11 int main()
12 {
13     srand(static_cast<unsigned int>(time(0)));
14
15     for (int count = 0; count < 2; ++count)
16         std::cout << randDam(1, 6);
17 }
```

### [Ответить](#)



1. *Onium:*

[25 июня 2020 в 20:58](#)

Нужно сбросить rand(). Как написано в последнем абзаце урока.

```
1 #include <iostream>
2 #include <cstdlib>
3 #include <ctime>
4
5 int randDam(int min, int max)
6 {
7     static const double fraction = 1.0 / (static_cast<double>(RAND_MAX) + 1.0);
8     return static_cast<int>(rand() * fraction * (max - min + 1) + min);
9 }
10
11 int main()
12 {
13
14     srand(static_cast<unsigned int>(time(0)));
15     randDam(rand(), rand()); // не знаю правилен ли такой сброс, но помог.
16
17     for (int count = 0; count < 1; ++count)
18         std::cout << randDam(1, 6);
19
20     return 0;
21 }
```

21 | }

[Ответить](#)

8. Денис:

[1 мая 2019 в 19:08](#)

Функция для вывода случайных чисел! Работает хорошо, результат не повторяется! можно не делать 2 переменные для цикла, я сделал для оформления, но так даже надёжней (не точно) 😊  
 P.S. Если использовать "srand(static\_cast<unsigned int>(time(0)))", результат в цикле очень схожий.

```

1  cout << "Введите минимальное случайное число: ";
2      int rmin;
3      cin >> rmin;
4      cout << "\nВведите максимальное случайное число: ";
5      int rmax;
6      cin >> rmax;
7      cout << "\nВведите количество чисел: ";
8      int cicle; //количество необходимых результатов
9      cin >> cicle;
10     int cicle_num = 1; //счётчик выдаваемых результатов
11     while (cicle > 0)
12     {
13         srand(time(0)*(cicle_num - cicle)); //создание условия для генератора ранд
14         int result = GetRandomNum(rmin, rmax); //Получаем значение функции генера
15
16         //чтобы выглядело всё аккуратно :)
17         if (cicle_num >= 1 && cicle_num < 10)
18             cout << cicle_num << " - ";
19         if (cicle_num >= 10 && cicle_num < 100)
20             cout << cicle_num << " - ";
21         if (cicle_num >= 100 && cicle_num < 1000)
22             cout << cicle_num << " - ";
23         if (cicle_num >= 1000 && cicle_num < 10000)
24             cout << cicle_num << " - ";
25         if (cicle_num >= 10000 && cicle_num < 100000)
26             cout << cicle_num << " - ";
27         cout << result;
28         if (result >= 10000)
29             cout << " |";
30         if (result >= 1000 && result < 10000)
31             cout << " |";
32         if (result >= 100 && result < 1000 )
33             cout << " |";
34         if (result >= 10 && result < 100)
35             cout << " |";
36         if (result < 10)
37             cout << " |";

```



```

38 //ограничение: 5 результатов на строку
39 if ((cicle_num)%5 == 0)
40     cout << "\n";
41     cicle_num += 1; //прибавляем +1 к счётчику результатов
42     cicle -= 1; // Отбавляем счётчик остатка циклов
43
44 }
45 system("pause");
46 }

```

[Ответить](#)

1. Борис:

[30 апреля 2020 в 20:53](#)

1) Что за "GetRandomNum"?

2) Выравнивание для аккуратности реализовано похабно. Можно сделать гораздо проще и лаконичнее (подсказка: использовать функцию длины строки, хотя есть и другие варианты).

[Ответить](#)

9. Вячеслав:

[15 марта 2019 в 19:39](#)

Вот так у меня получилась прога с случайным броском кубика:

```

1 #include "pch.h"
2 #include <iostream>
3 #include <cstdlib>
4 #include <ctime>
5 // Генерируем случайное число между значениями min и max
6 // Предполагается, что функцию srand() уже вызывали
7 int getRandomNumber(int min, int max)
8 {
9     static const double fraction = 1.0 / (static_cast<double>(RAND_MAX) + 1.0);
10    // равномерно распределяем случайное число в нашем диапазоне
11    return static_cast<int>(rand()* fraction *(max - min + 1) + min);
12 }
13
14 int main()
15 {
16     using namespace std;
17    // устанавливаем значение системных часов в качестве стартового числа
18    srand(static_cast<unsigned int>(time(0)));
19    //выставляем параметры цикла, чтобы вывело 10 чисел
20    for (int count = 0; count < 10; ++count)
21        //вывод диапазон от 1 до 6 с отступом в 1 таб
22

```

```

23         cout << getRandomNumber(1, 6) << "\t";
24
25     return 0;
    }

```

[Ответить](#)10. *Dagon:*[17 февраля 2019 в 20:51](#)

Как сделать диапазон для Мерсенна?! Спасибо.

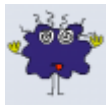
[Ответить](#)11. *Oleksiy:*[23 августа 2018 в 12:13](#)

"Вот небольшая функция, которая конвертирует результат функции rand() в нужный нам диапазон значений:"

Не понятно, как происходит конвертация. Там идет округление конечного результата? Когда в знаменателе стоит выражение:

$(\text{static\_cast<double>}(\text{RAND\_MAX}) + 1.0)$

то при  $\text{rand()} = \text{RAND\_MAX}$  выражение  $(\text{rand()} * \text{fraction} * (\text{max} - \text{min} + 1) + \text{min})$  не даст "чистую" шестерку. Этот алгоритм основан на том, что  $\text{int max}$  намного меньше  $\text{RAND\_MAX}$ ?

[Ответить](#)12. *Эдуард:*[11 августа 2018 в 19:12](#)

"Однако есть простое решение: вызовите функцию rand() один раз и сбросьте результат. Затем вы сможете использовать rand() как обычно в вашей программе".

Не понимаю как реализовать сброс результата.

[Ответить](#)1. *Юрий:*[21 августа 2018 в 21:35](#)

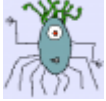
Просто вызвать rand() в начале функции main():

```

1  rand();

```

А затем, когда уже будет нужно сгенерировать число — опять вызвать rand(). Таким образом у вас будет 2 вызова rand(), первый из которых будет выполнять сброс.

[Ответить](#)13. *Алексей:*[28 марта 2018 в 16:12](#)

А что такого плохого в генерации числа близкого к начальному в Visual Studio, оно все равно генерируется от миллионов секунд. Обычный пользователь вряд ли будет их высчитывать.

[Ответить](#)1. *Юрий:*[29 марта 2018 в 18:21](#)

Вы правы, обычный пользователь его вычислять не будет и в тривиальных программах сброс первого значения можно упустить. Но если вы разрабатываете серьезное программное обеспечение/софт/игры или хотите/будете в будущем разрабатывать что-либо подобное, где рандомизация значений будет играть важную роль, то ознакомиться с этим нюансом и способом его решения — нелишне, не так ли?

[Ответить](#)14. *Герман:*[31 января 2018 в 19:50](#)

С математикой все ясно. Не совсем ясен алгоритм работы программы:

```
1 #include <iostream>
2
3 unsigned int PRNG()
4 {
5     // наше стартовое число - 4541
6     static unsigned int seed = 4541;
7
8     // Берем стартовое число и, с помощью него, генерируем новое значение
9     // Из-за использования констант с большими значениями и переполнения, будет
10    // очень трудно угадать следующее число
11    // исходя из предыдущего.
12    seed = (8253729 * seed + 2396403);
13
14    // Берем стартовое число и возвращаем значение в диапазоне от 0 до 32767
15    return seed % 32768;
16 }
17
18 int main()
19 {
20     // Выводим 100 случайных чисел
21     for (int count=0; count < 100; ++count)
```

```
22 {  
23     std::cout << PRNG() << "\t";  
24  
25     // Если вывели 5 чисел, то продолжаем с новой строки  
26     if ((count+1) % 5 == 0)  
27         std::cout << "\n";  
28 }  
29 }
```

Почему в каждой итерации цикла функция PRNG() возвращает разные значения?

[Ответить](#)

1.  Юрий:

[1 февраля 2018 в 01:46](#)

Так ведь в комментариях это объясняется. 4541, 8253729, 2396403, 32768 — это большие числа и переменная seed типа unsigned int не сможет хранить столь большое число, которое получится после выполнения всех арифметических операций с seed — произойдет переполнение переменной и мы получим какое-то вообще левое число. И так будет со всеми 99-тью следующими числами. Алгоритм построен на переполнении переменной.

[Ответить](#)



1. Герман:

[1 февраля 2018 в 10:08](#)

То что алгоритм построен на переполнении переменной это понятно, не понятно другое, как влияет на значение переменной Static, ведь после каждой итерации цикла значение seed разное, а у переменной атрибут static- все 100 значений переменной становится одинаковое!

[Ответить](#)

1.  Юрий:

[1 февраля 2018 в 22:45](#)

Ключевое слово static делает переменную статической — её значение сохраняется за пределами блока, в котором она определена и используется. После выхода из блока её значение не уничтожается, а сохраняется. Область видимости такой переменной уже не локальная, а статическая. Смотрите [урок 51](#).



2. Дасту:

[4 октября 2018 в 10:15](#)

В 6 строчке происходит инициализация статической переменной, т.е. эта строчка будет выполняться только один раз за все время работы. Всякий раз, когда функция будет снова вызываться, seed будет иметь последнее значение, данное ему в строчке 12 игнорируя при этом 6 строчку.



15. *Герман:*

[31 января 2018 в 19:12](#)

```
1 unsigned int PRNG()  
2 {  
3     static unsigned int seed = 4541;  
4     seed = (8253729 * seed + 2396403);  
5     return seed % 32768;  
6 }
```

Уважаемый автор, не совсем ясно как работает инкремент переменной seed в данном фрагменте при итерациях цикла FOR.

[Ответить](#)



1. *Юрий:*

[31 января 2018 в 19:21](#)

Конкретизируйте ваш вопрос — что именно непонятно? Здесь выполнение арифметических операторов, не больше. Умножение, сложение и остаток.

[Ответить](#)



1. *Игорь:*

[22 июля 2018 в 06:30](#)

Может он имел в виду то, что если функция постоянно вызывается циклом то почему переменной seed заново не присваивается значение 4541. А как я понимаю, суть как раз в статической переменной, которая после первой инициации, второй раз уже её игнорит.

[Ответить](#)

## Добавить комментарий

Ваш E-mail не будет опубликован. Обязательные поля помечены \*






Имя \*

Email \*

Комментарий

☐ Сохранить моё Имя и E-mail. Видеть комментарии, отправленные на модерацию☐ Получать уведомления о новых комментариях по электронной почте. Вы можете [подписаться](#) без комментирования.[TELEGRAM](#)  [КАНАЛ](#)[ПАБЛИК](#) 

## ТОП СТАТЬИ

-  [Словарь программиста. Сленг, который должен знать каждый кодер](#)
-  [Урок №1. Введение в программирование](#)
-  [70+ бесплатных ресурсов для изучения программирования](#)
-  [Урок №1: Введение в создание игры «SameGame» на C++/MFC](#)
-  [Урок №4. Установка IDE \(Интегрированной Среды Разработки\)](#)

- [Ravesli](#)
- - [О проекте/Контакты](#) -
- - [Пользовательское Соглашение](#) -
- - [Все статьи](#) -
- Copyright © 2015 - 2020