Поднимаем рандом на имеющимся софте.

Оглавление

Введение

Сегодня использование случайных чисел в программах – не редкость. Криптография, компьютерные игры, симуляторы и компьютерное моделирование невозможны без случайных чисел.

В криптографии случайные числа используются в качестве ключей шифрования. В компьютерных играх случайные числа определяют «случайности» происходящие в игре, а также являются источником «принятия решений» ботов. Нередко случайные числа влияют на сложность игры. В компьютерном моделировании случайные числа необходимы для имитации хаотичных процессов.

Связи с этим у человечества существует неразрешимая задача – создать генератор абсолютно случайных чисел, в том числе и для компьютеров. Частичным решением данной проблемы является увеличение случайности уже существующих генераторов путем накопления случайности.

Случайные числа

Существует большое количество заумных определений понятия «случайное число», но давайте отойдем от практики «умное и непонятное» к практике «как для простых».

В первую очередь, случайные числа – это последовательность чисел. При этом, числа должны «браться» не абы как, а из какого-нибудь диапазона, например от 1 до 10, или от 0 до бесконечности. Как правило, диапазон выбирает сам человек в зависимости от того, что необходимо реализовать в программе. Сами же числа имеют ряд свойств, которые также подбираются человеком при написании программы. Хорошим примером является наличие у чисел дробной части: в одних задачах удобнее использовать только целые числа, в других только с дробной частью. Иногда удобнее использовать одновременно и те и те. Весьма немаловажным является условие наличия или отсутствия знака у генерируемых чисел, ведь в большом количестве задач отрицательные числа использовать нельзя.

Очень важно, чтобы в между всеми числами последовательности не было статистической зависимости, например, каждое 2 число – 1, или каждое 5 число – 2. Это позволяет предсказывать какое число будет следующим, а значит, последовательность не случайна. Должна быть исключена и математическая зависимость – каждое последующее число не должно создаваться по математической формуле от предыдущих чисел. Наличие такой зависимости также позволяет предсказать следующее число. Хорошим примером математической зависимости может служить последовательность чисел 2, 4, 8, 16. Вы сразу можете сказать, что следующее число будет 32, т.к. все числа формируются умножением предыдущего числа на 2. Это не есть случайные числа. Метод, по которому «выбираются» случайные числа называется законом распределения.

Итак, соберем все это воедино и получим определение понятия «случайные числа»: случайные числа – это последовательность чисел, которая составлена из чисел определенного диапазона, между которыми нет никакой статистической и математической зависимости, и подчиняется какому-либо закону распределения.

Генераторы случайных чисел

Для генерации случайных чисел человек прибегает к различным предметам, которые могут создавать (генерировать) случайные числа. Самым простым примером генератора случайных чисел является игральный кубик. Бросая кубик, мы получаем какое-нибудь число от 1 до 6. Другими словами, мы получаем число из диапазона от 1 до 6. При этом функцией распределения здесь являются законы физики – они действуют на кубик так, что нельзя предсказать, как он упадет (какое число выпадет).

Не менее известным генератором случайных чисел является монета. Подбрасывая монету, мы случайно получаем «орел» либо «решку». В этом случае мы получаем значение из диапазона от 1 («орел») до 2 («решка»).

Законом распределения здесь также являются законы физики, от которых зависит как упадет монета (что на ней выпадет).

Если Вы играли когда либо в карты, то вы сталкивались с необходимостью перетасовки колоды («перемешиванию» карт, чтобы они были в колоде распределены случайным образом). Здесь, вытягивая карту из колоды, получаем значение от 2 до туза (в стандартной колоде), а законом распределения является Ваша рука, которая перетасовала карты и выбрала одну.

Шансы и вероятность случайного числа

Бросая кубик или монету, «перемешивая» карты, мы получаем какое либо число (или значение) из определенного диапазона. У игрального кубика диапазон равен 5 (от 1 до 6). 6 мы можем получить только если выпадет грань с 6, которая только одна у кубика. Это значит, мы можем получить 6 только 1 способом. Если у кубика 6 была на двух гранях, это означало бы , что мы можем получить 6 двумя способами.

Зная количество способов получения конкретного результата и число возможных результатов всего, мы можем вычислить шанс выпадения конкретного числа.

Каков шанс, что мы, при броске игрального кубика, получим 6? Всего способов получить 6 у нас один, т.к. она только на 1 грани кубика, а всего вариантов, которые могут выпасть тоже 6: 1, 2, 3, 4, 5 и 6. Таким образом, наши шансы равны 1 к 6 (1:6).

Каков шанс, что при подбрасывании монеты мы получим «орел»? Всего способов получить «орла» 1, т.к. он есть только на одной стороне монеты. Всего вариантов у нас 2 («орел» или «решка»), поэтому шансы выпадения орла 1 к 2 (1:2).

Каков шанс вытащить туза из колоды карт? Всего тузов в колоде 4, таким образом, мы можем вытащить туз 4 способами. При этом, всего возможных

результатов вытаскивания карты 52 (в колоде 52 карты). Таким образом, вероятность вытащить туза 4 к 52 (4:52).

Зная шансы получения какого-либо значения, мы можем вычислить вероятность выпадения этого числа. Вероятностью мы называем уровень возможности наступления какого-либо события. Вычисляется вероятность события очень просто – нужно число способов, которыми можем получить нужный нам результат, на число всевозможных результатов:

$$P = \frac{n}{m}$$

где: P – вероятность появления какого-либо конкретного значения , n – количество шансов получить нужное значение, m – число всевозможных значений.

Так, вероятность выпадения «орла» на монете равна 1:2=0,5. Вероятность выпадения 6 на кубике 1:6≈1,67 . Вероятность вытащить туза из колоды карт 4:52=1:13≈0,77.

Шансы и вероятность в идеальных и неидеальных генераторах

В идеальных генераторах случайных чисел каждое из возможных чисел может выпасть только 1 способом. Таким образом, у всех идеальных генераторов все возможные варианты чисел имеют одинаковые шансы на выпадение (равновероятны). Так, любое число на кубике может выпасть одним способом, т.е. шансы у любого из них на выпадения одинаковые – 1:6 (вероятность ≈1,67). «Орел» и «решка» также могут выпасть только 1 способом, поэтому их шансы 1:2 (вероятность = 0,5). У любой карты из колоды шанс быть вытащенной 1:52 (вероятность ≈0,19).

К сожалению, в реальной жизни идеальных генераторов случайных чисел почти нет. Так, у монеты, на разных сторонах разный рисунок, что приводит к тому, что одна сторона тяжелее другой. Учитывая, что монета чаще будет падать тяжелой стороной вниз, чаще всего будет выпадать легкая сторона. Таким образом, выпадения «орла» и «решки» имеют разные вероятности (у

одного больше шансов выпасть, чем у другого). Такой генератор нельзя назвать случайным.

Такая же ситуация и с кубиком. У него разные грани имеют разную массу, в результате чего одни числа выпадают чаще, чем другие. Это тоже не случайный генератор.

ГПСЧ

Компьютер – это машина, устройство. В нем все спланировано, все задумано. В нем нет ничего случайного. Связи с этим, в компьютере неоткуда браться случайным числам. Они не могут их создавать.

Вместо случайных чисел компьютеры используют псевдослучайные числа – последовательности чисел, которые выгладят как случайные, но таковыми не являются. Генераторы таких чисел называются генераторами псевдослучайных чисел (ГПСЧ). Вероятности появления чисел, у данных генераторов, не одинаковые. Одни числа имеют больше шансов выпасть, чем другие.

Измеряем «не случайность»

У любого ГПСЧ есть различные параметры, которые можно измерить. Самыми важными из всех параметров ГПСЧ являются уровень случайности ГПСЧ и энтропия создаваемых чисел. Чтобы убедится, что ГПСЧ создает числа с большой случайностью, нам необходимо измерить его случайность. Это возможно только статистическими методами. Мы можем рассчитать вероятность появления каждого из возможных чисел ГПСЧ. Как Вам уже известно, она измеряется числом, которое находится в диапазоне от 0 (невозможное событие) до 1 (событие, которое точно произойдет). При этом в идеальных генераторах случайных чисел все возможные числа равновероятны, а в ГПСЧ – нет. В идеальных генераторах наиболее вероятное и наименее вероятное числа имеют одинаковую вероятность.

$$Вероятность\ числа\ A = Вероятность\ числа\ B$$

$$P(A) = P(B)$$

Чтобы найти «не случайность» ГПСЧ, нам необходимо найти наиболее и наименее вероятные числа, которые он может генерировать, а затем из первого вычесть второе.

$$Не\ случайность = \begin{matrix} вероятность \\ наибюолее \\ вероятного \\ числа \end{matrix} - \begin{matrix} вероятность \\ наименее \\ вероятного \\ числа \end{matrix}$$

Обозначим «не случайность» буквой Q, а вероятности наиболее вероятного и наименее вероятного чисел P_{max} и P_{min} мы получим следующую формулу:

$$Q = P_{max} - P_{min}$$

Так, если вы заставите идеальный генератор случайных чисел создать большое количество случайных чисел от 0 до 9, то шансы появления каждого из этих чисел будет одинаковы и равны 1:10 (вероятность появления 0,1). Т.е наиболее вероятное и наименее вероятное числа будут иметь одинаковую вероятность 0,1. «Не случайность» такого генератора равна 0, т.к. 0,1-0,1=0. Это значит, что все числа, создаваемые этим генератором, имеют «не случайность» равную 0, т.е. они абсолютно случайны.

Теперь давайте рассчитаем «не случайность» для абсолютно неслучайного генератора. Это означает, что каждый раз он создает одно и то же число. Шансы появления такого числа равны 10:10 (вероятность появления 1/1=1). При этом, так как другие числа появится не могут, их шансы на появление 0:10 (вероятность появления 0/10=0). Теперь, по формуле «не случайности», найдем «не случайность» данного ГПСЧ: 1-0=1. Таким образом, данный генератор имеет «не случайность» 1, что говорит о том, что он абсолютно неслучаен! Это значит, что все числа, созданные данным генератором, не имеют обсалютно предсказуемы.

Обратите внимание! «Не случайность», как и вероятность, измеряется числом в диапазоне от 0 до 1. 0 – это абсолютно случайный генератор, а 1 – абсолютно неслучайный генератор.

Давайте рассчитаем «не случайность» для ГПСЧ. Представим, что наш ГПСЧ может случайным образом выдать только 0 или 1. Как рассчитать вероятность 0 и 1, если они не равновероятны? Только путем статистики: заставить ГПСЧ создать большое количество случайных чисел, а затем воспользоваться нашей формулой вероятности. Допустим, мы запустили наш ГПСЧ и он сгенерировал 10 чисел, из которых 4 являются единицами, а 6 нулями. Шансы появления единицы 4:10, а шансы появления нуля 6:10. Теперь посчитаем вероятности этих чисел. Вероятность появления единицы равна 4/10=0,4. Вероятность появления нуля равна 6/10=0,6. Теперь от вероятности наиболее вероятного числа (нуля) отнимем вероятность наименее вероятного числа (единицы): 0,6-0,4=0,2. «Не случайность» нашего ГПСЧ равна 0,2. Это означает, что все эти числа имеет «не случайность», равную 0,2.

Давайте для более полного понимания разберем более сложный пример. Допустим, наш ГПСЧ создает случайные числа от 0 до 9. Для наиболее лучшего результата заставим его создать миллион случайных чисел и посчитаем, сколько и каких чисел он создал. Затем укажем шансы появления каждого из возможных чисел и посчитаем их вероятности. Для удобства я это свел в таблицу:

Число	Количество появлений	Шансы на появление	Вероятность
0	100115	100115:1000000	0,100115
1	99970	99970:1000000	0,09997
2	100100	100100:1000000	0,1001
3	100283	100283:1000000	0,100283
4	100099	100099:1000000	0,100099
5	99782	99782:1000000	0,099782
6	100159	100159:1000000	0,100159
7	99824	99824:1000000	0,099824
8	100220	100220:1000000	0,10022
9	99448	99448:1000000	0,099448

Найдем из этой таблицы наиболее и наименее вероятные числа. Наиболее вероятным числом является тройка (ее вероятность 0,100283). Наименее вероятным числом является девятка (ее вероятность 0.099448).

Чтобы рассчитать «не случайность» нам нужно от вероятности наиболее вероятного числа (тройки) отнять вероятность наименее вероятного числа (единицы): 0.100283-0.099448=0,000835. Не случайность нашего ГПСЧ равна 0,000835, что гораздо меньше «не случайности» ГПСЧ из предыдущего примера. Это говорим о том, что этот ГПСЧ более случаен, чем предыдущий.

Вычисляем случайность по «не случайности»

Вычислив «не случайность» ГПСЧ, мы можем вычислить случайность. Дело все в том, что случайность, как и «не случайность» с вероятностью измеряется от 1 (абсолютно случайно) до 0 (абсолютно неслучайно). Таким образом, если «не случайность» ГПСЧ равна 0 (ГПСЧ идеальный), то его случайность равна 1. И наоборот, если «не случайность» равна 1 (ГПСЧ не случайный), то случайность равна 0. Таким образом, случайность обратная сторона «не случайности». Эта взаимосвязь дает нам возможность измерить случайность, просто отняв «не случайность» от 1:

$$S = 1 - Q$$

Давайте рассчитаем случайность ГПСЧ из уже приведенных выше 4х примеров.

В первом нашем примере «не случайность» была равна 0, т.к. мы использовали идеальный генератор случайных чисел. Таким образом, случайность данного генератора равна 1-0=1. Он абсолютно случаен.

Во втором примере «не случайность» была равна 1. Таким образом, случайность данного генератора равна 1-1=0. Он абсолютно не случаен.

В третьем примере мы получили значение «не случайности» равное 0,2. Получается случайность данного ГПСЧ будет 1-0,2=0,8.

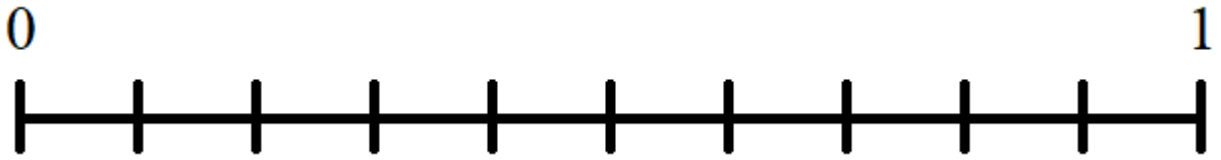
В четвертом примере «не случайность» была равна 0,000835. Вычислим его случайность: 1-0,000835=0,999165.

Случайность ГПСЧ из 1 примера меньше, чем ГПСЧ из 2 примера. Все сходится, второй генератор более случаен.

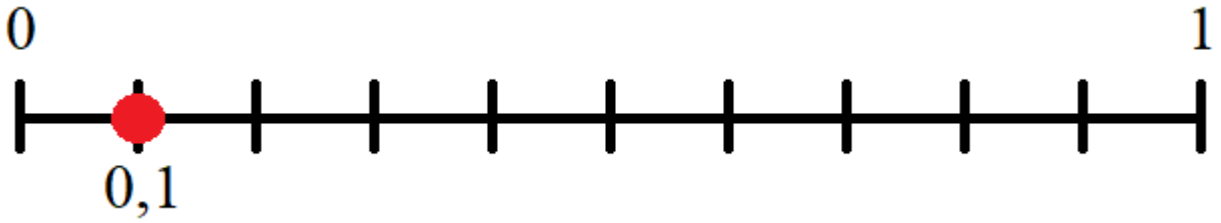
Наглядное объяснение вышеописанных измерений и вычислений

Людям, не сталкивающимся ранее с теорией вероятности, может быть довольно сложно воспринять выше описанную информацию. Связи с этим, считаю полезным дать объяснение всем этим формулам и измерениям более наглядно – при помощи координатного отрезка.

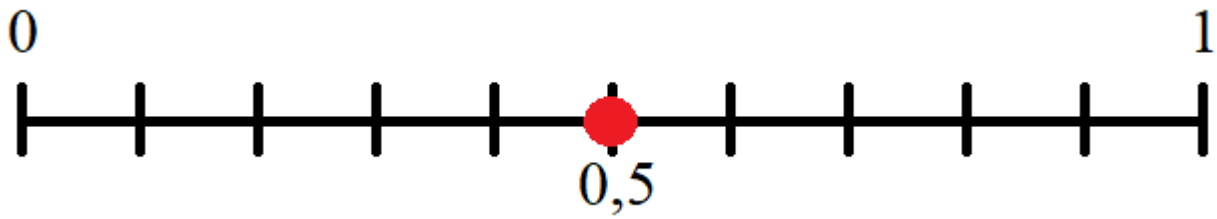
Представим вероятность как координатный отрезок с минимальной координатой 0, а максимальной 1:



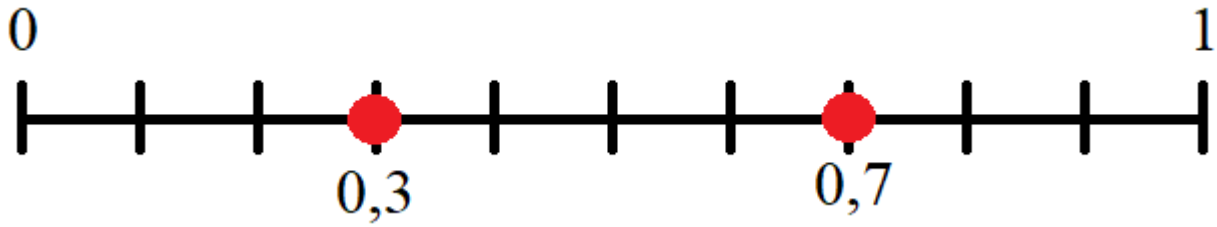
В идеальном генераторе случайных чисел вероятность появления всех чисел одинакова. Таким образом, если мы отметим точкой на этом отрезке вероятности всех чисел, то они попросту наложатся друг на друга и нам будет казаться, что это одна точка. Так, например, в случае генерации случайных чисел от 0 до 9, то все числа от 0 до 9 будут иметь вероятность появления 1/10=0,1. Вот так это будет выглядеть:



Если же мы кидаем идеальную монетку, у которой может выпасть «орел» или «решка», вероятность появления у «орла» и «решки» 1/2=0,5. Выглядеть это будет вот так:



В случае с ГПСЧ, вероятности появления разных чисел разные. Допустим, ГПСЧ выдает только два числа (0 или 1), причем вероятности появления у этих чисел следующие: у нуля 0,3, а у единицы 0,7. Естественно, на координатном отрезке это будет 2 точки:



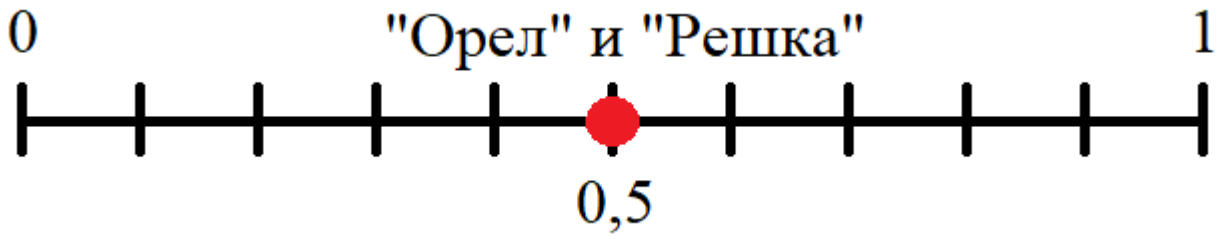
«Не случайность» - это то, в пределах какого диапазона вероятности чисел отличаются друг от друга. Для наглядности я выделю этот диапазон желтым цветом:

Как определить размер этого диапазона? Правильно, от 0,7 отнять 0,3. Т.е. от наибольшей вероятности отнять наименьшую. 0,7-0,3=0,4. Значит «Не случайность» (желтый диапазон) равен 0,4. Т.е. мы пришли к нашей формуле:

$$Q = P_{max} - P_{min}$$

Давайте рассчитаем несколько примеров.

В случае использования идеальной монеты мы получаем одинаковые вероятности появления у орла и решки:

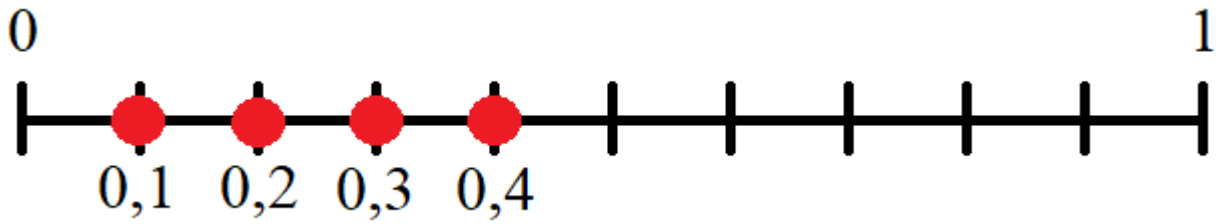


Каков размер желтого диапазона («Не случайности») ? Правильно, его нет, т.е. 0. Данный генератор случайных чисел идеальный.

Разберем пример посложнее. Воспользуемся ГПСЧ, который генерирует псевдослучайные числа от 0 до 3 со следующими вероятностями:

Число	Вероятность
0	0,1
1	0,2
2	0,3
3	0,4

Отобразим это на координатном отрезке:



В этом диапазоне расположены вероятности всех наших чисел:

Таким образом, в нашем случае, «не случайность» равна 0,4-0,1=0,3.

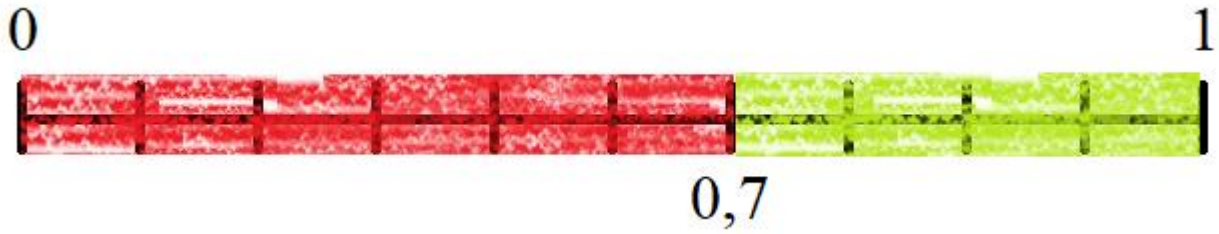
Теперь подобным образом представим отношение случайности и «не случайности». Учитывая, что «не случайность» обратная величина

случайности, и колеблятся они в диапазоне от 0 до 1, их взаимоотношение можно представить в виде шкалы.

Если генератор идеальный, то вся шкала зеленая:

Если генератор абсолютно неслучайный, то шкала абсолютно красная:

Представим, что у нашего ГПСЧ «не случайность» равна 0,7. Графически это выглядит вот так:



Теперь представим, что у нашего ГПСЧ «не случайность» равна 0,1:

Каким образом нам, зная красное («не случайность») найти зеленое (случайность)? Правильно, от 1 отнять красное. Иными словами мы пришли к следующей формуле:

$$S = 1 - Q$$

Т.е. в первом случае «зеленое» (случайность) равна 1-0,7=0,3. Во втором случае случайность равна 1-0,1=0,9.

Накопление случайности

Если мы возьмем известное число и умножим его на случайное число, то в результате мы получим случайное число. Не смотря на то, что первое слагаемое было не случайным, результат все равно получился случайным. Иными словами, случайность «унаследовалась» от второго числа, которое было случайным. Произведение несет ту случайность, которая была заложена в случайное число, таким образом, мы сохранили случайность. Вполне логично, что если мы сомножители поменяем местами (умножим случайное число на неслучайное число), мы все равно получим случайное число. Иными словами, мы также сохраним случайность. Выводом из этого является тот факт, что случайность не может быть уменьшена – «наследуется».

Если умножить случайное число на другое случайное число, то мы получим чсло, более случайное чем сами сомножители. Иными словами, мы «унаследовали» и «накопили» случайность от двух случайных чисел.

Думаю, стоит прояснить общий механизм наследования случайности. Дело все в том, что совершая математическое действие над двумя числами мы «создаем» одно число. Иными словами, два числа, над которыми проводится действие, «смешиваются» в одно число. В результате «перемешки» неслучайного и случайного чисел, итоговое число наделяется случайностью, которой обладало случайное число. Оно передает свою случайность итоговому числу. В случае если оба числа случайные, то они оба передают свою случайность итоговому числу, тем самым суммируя случайности. Иными словами происходит накапливание.

Учитывая, что любое математическое действие над двумя числами приводит к их «смешиванию», накопление случайности возможно на любых математических действиях, а не только на умножении, как в примерах выше.

Эффективность методов «перемешивания»

Как уже было описано выше, «перемешивание» чисел и накопление случайности происходит на любых математических действиях. Однако, стоит отметить, что накопление случайности на одних действиях происходит куда лучше, чем на других. Объясню почему на конкретном примере:

ГПСЧ сгенерировал два числа, допустим 862 и 4. Сложив эти два числа, мы получим 867 (862+4=867). Обратите внимание на то, что поменялся лишь 1 разряд (2 на 7), другие два остались без изменений. Однако если мы применим вместо сложения умножение, то получим 3448 (862*4=3448). Мы изменили не только все разряды, но даже изменили и их число! Таким образом, случайности накопилось куда больше. Думаю, Вам уже понятно, что возведение в степень будет накапливать случайность еще эффективнее.

Исходя из этого, мы приходим к вопросу о том, какой же способ перемешки наиболее эффективный.

Наиболее эффективным методом пересмешки является перемешка числа по разрядно. Иными словами единицы с единицами, десятки с десятками и т.д. Кроме того, чем элементарнее будет представлено число, тем лучше будет «перемешивание». Самым элементарным представлением числа будет его представление в двоичной системе счисления.

Иными словами, самое эффективное накопление случайности происходит при «перемешивании» двоичных чисел, выполняя над ними какие-либо действия побитно. Наиболее простым вариантом такого «перемешивания» является использование побитных логических функций.

Побитовых логических функций много, однако, в языках программирования, встречаются далеко не все функции. Во всех языках программирования встречаются лишь 3 функции: «и», «или» и «сложение по модулю 2». Чаще всего оно обозначаются словами «AND», «OR» и «XOR».

Давайте представим таблицу истинности данных функций и найдем наиболее подходящую для накопления случайности:

AND		
A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

OR		
A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

XOR		
A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

При использовании AND чаще всего в будет получаться 0. Таким образом, при «смешивании» чисел этой функцией мы будем получать числа в двоичном формате, которые будут иметь нулей в 3 раза больше. Это негативно влияет на случайность чисел. Аналогичная ситуация с OR: при ее использовании мы будем получать 1 в три раза чаще, чем 0. А вот с XOR – совсем другая ситуация: при ее использовании вероятность получения 0 и 1 одинаковы. Она наиболее эффективна.

Самый лучший способ накапливать случайность – это использование побитного XOR к двум случайным числам.

Измерение накопления случайности и средний коэффициент накопления

На практике может возникнуть необходимость измерять не только случайность ГПСЧ, но и количество случайности, накопленной при помощи XOR (ΔS). Сделать это возможно только при помощи разности случайностей, полученных экспериментальным путем. Так мы можем вычислить случайность при смешивании двух и трех случайных чисел, а затем взять их разность. В виде формулы это можно выразить так:

$$\Delta S = S_{n+1} - S_n$$

Приведем пример: при смешивании двух случайных чисел мы получили случайность, равную 0,8. При смешивании трех случайных чисел мы получили случайность 0,9. Таким образом, мы накопили 0,9-0,8=0,1 случайности.

Стоит отметить, что ΔS может быть как положительной, так и отрицательной величиной. Может произойти так, что при смешивании трех чисел мы получили случайность 0,9, а при четырех 0,7. Получается, что мы накопили 0,7-0,9=–0,2 случайности.

Данное значение не является константой и постоянно меняется, так как мы имеем дело со случайными числами. Данное значение служит лишь индикатором того, удалось ли нам увеличить случайность. Так, если ΔS больше 0, мы увеличили случайность. Если ΔS, наоборот, меньше 0, то мы уменьшили случайность. Если ΔS равно 0, то уровень случайности не изменился.

Измеряя накопленную случайность мы можем сравнить различные ГПСЧ и узнать, на каком из них лучше удается накапливать случайность. Для этого мы должны вычислить средний коэффициент накопления (среднюю скорость накопления) случайности (k). Сравнив коэффициенты двух ГПСЧ, мы сможем определить на каком из них лучше удается накопление случайности.

Средний коэффициент накопления случайности – это среднее арифметическое всех ΔS. В виде формулы это выражается так:

$$k = \frac{\Delta S_1 + \Delta S_2 + \cdots + \Delta S_n}{n}$$

Приведем пример. У нас есть два ГПСЧ, назовем их ГПСЧ 1 и ГПСЧ 2. Проведем ряд экспериментов и вычислим по три ΔS для каждого из них:

ГПСЧ 1		
XOR	Параметры	
	S	ΔS
0	0,7	
1	0,8	+0,1
2	0,9	+0,1
3	0,10	+0,1

ГПСЧ 2		
XOR	параметры	
	S	ΔS
0	0,9	
1	0,8	–0,1
2	0,84	+0,01
3	0,99	+0,15

Найдем k для ГПСЧ 1: для этого найдем среднее арифметическое всех ΔS:

$$\frac{0,1 + 0,1 + 0,1}{3} = \frac{0,3}{3} = 0,1$$

Теперь найдем k для ГПСЧ 2:

$$\frac{(-0,1) + 0,1 + 0,15}{3} = \frac{0,15}{3} = 0,05$$

Сравнив k мы увидим, что у ГПСЧ 1 он больше. Следовательно, на этом ГПСЧ накопление случайности идет быстрее.

О практических требованиях во время генерации случайных чисел

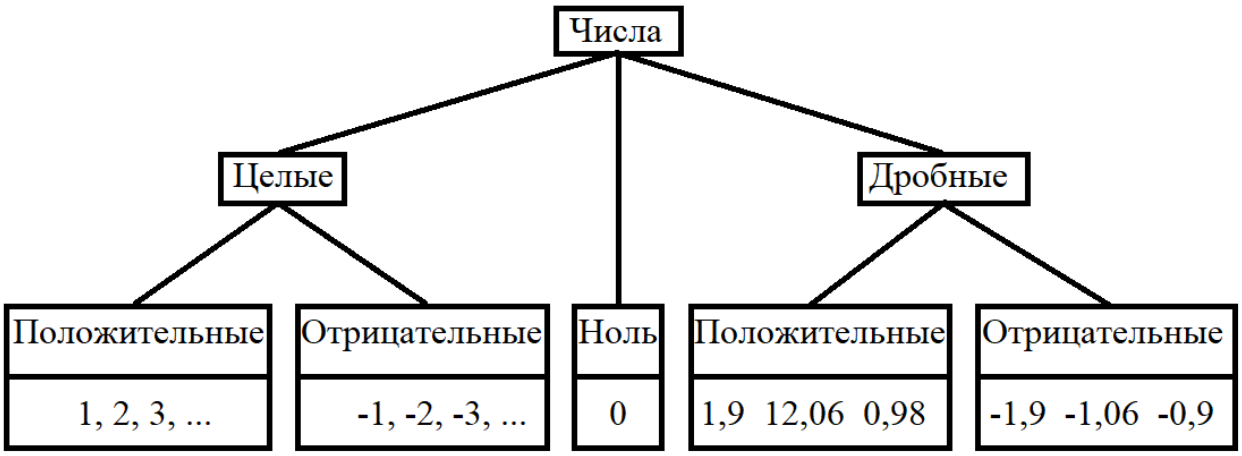
На практике генерация случайных чисел зачастую усложняется какими-либо требованиями. Чаще всего встречаются ограничения по диапазонам, форматам и размеру генерируемых чисел.

Ограничение по диапазону заключается в том, что для решения какой-либо задачи, требуется генерировать случайные числа в каком-либо диапазоне, например от 1 до 9, или от 0 до 100. Таким примером может быть имитация бросания игрального кубика на компьютере: ГПСЧ должен генерировать случайное число в диапазоне от 1 до 6. Для соблюдения данных условий в некоторые ГПСЧ встроены функции их «настройки». Так, в языке PHP, ГПСЧ «rand» имеет возможность регулировать диапазон, указывая в скобках минимальное и максимальное число через запятую.

Диапазон	В команде rand
От 0 до 10	rand(0, 10);
От 0 до 9	rand(0, 9);
От 1 до 6	rand(1, 6);
От 25 до 100	rand(25, 100);
От 10 до 12	rand(10, 12);
От A до B	rand(A, B);

Ограничения генерируемых чисел по формату подразумевает тип генерируемых чисел. Числа могут быть целыми и дробными,

положительными и отрицательными. Для простоты представим это в виде изображения:



В разных задачах нужно использовать разные типы чисел. Так, при имитации игрового кубика нужно генерировать целые положительные числа. При генерации случайных координат в декартовой системе чаще всего приходится использовать целые положительные и отрицательные числа. При моделировании физических процессов приходится использовать дробные положительные числа, а в механике зачастую вместе с дробными отрицательными.

Такие требования к задачам приводят нас к тому, что нужно знать какой ГПСЧ мы используем, чтобы генерировать псевдослучайные числа нужного типа. Или же уметь приводить их к нужному виду.

Довольно часто встречаемым критерием является размер генерируемого числа, т.е. количество цифр в нем. Так, при генерации кода к банковской карте, вам придется генерировать четырехзначные числа, а при генерации серии и номера паспорта десятизначное число (4 цифры серия, 6 цифр номер).

Приведение сгенерированных чисел к нужным параметрам

Используя различные ГПСЧ вы столкнетесь с необходимостью приведения чисел к нужному формату. Как уже было сказано, это делается при помощи математических действий над сгенерированным случайным числом.

В первую очередь стоит отметить, что числа в компьютере делятся на целые и дробные. Перевод числа из одного типа в другой и обратно производится специальными командами в языках программирования, поэтому рассматривать это в данной статье мы не будем.

Некоторые ГПСЧ генерируют не только положительные, но и отрицательные числа. В некоторых задачах это может стать проблемой. Так, например, эмитируя игральный кубик, мы можем выдавать только положительные числа. Чтобы ГПСЧ выдавал положительные числа, необходимо брать сгенерированные им числа в модуле:

$$|a| = a$$
$$|-a| = a$$

Если нам необходимо генерировать только отрицательные числа, то мы должны модуль случайного числа умножить на -1. Модуль числа делает как отрицательное, так и положительное число, положительным числом. Затем, умножая положительное число на -1 мы делаем его отрицательным:

$$|a| \cdot -1 = a \cdot -1 = -a$$
$$|-a| \cdot -1 = a \cdot -1 = -a$$

Наиболее сложной задачей является приведение генерируемых чисел к необходимому диапазону. У диапазона две границы – верхняя (самое большое число) и нижняя (самое маленькое число). Верхний диапазон регулируется взятием остатка от деления (обозначается «mod»), а нижний сложением.

Взятие остатка от деления – это когда одно число делится на другое с остатком, только в результате указывается не частное, а сам остаток. Примеры:

2 mod 5= 2 потому что 2/5=0 (остаток 2).

100 mod 15=10 потому что 100/15=6 (остаток 10)

18 mod 2=0 потому что 18/2=9 (остаток 0).

Функция взятия остатка в разных языках программирования обозначается по разному, например знаком процента (%) в языках программирования C, C++, C#, PHP и Java, а вот в языках Pascal и BASIC оно обозначается словом «MOD». Как Вы могли уже заметить, остаток от деления всегда меньше делителя. Какое бы число мы не разделили на делитель, остаток всегда будет меньше делителя. Именно это и используется при указании верхнего диапазона.

Стоит отметить, что используя функцию mod, для создания верхней границы диапазона, мы получаем диапазон от 0 до самого ближайшего, но меньше делителя, числа:

$$x \ mod \ n = c$$
$$0 \leq c < n$$

Приведем несколько примеров:

0 mod 5=0	3 mod 5 = 3	6 mod 5 = 1	9 mod 5 = 4
1 mod 5=1	4 mod 5 = 4	7 mod 5 = 2	10 mod 5 = 0
2 mod 5 = 2	5 mod 5 = 0	8 mod 5 = 3	

Как видите, самый большой остаток это 4, а делили мы на 5 (делитель, n).

После установления верхней границы нужно установить нижнюю. Как уже было сказано, нижняя граница устанавливается сложением и вот почему: если к какому-либо числу прибавить случайное число (которое должно быть положительным) мы получим случайное число больше или равное тому, к которому прибавляли.

$$a + x = c$$
$$c \geq a$$

Вот несколько примеров:

5+0=5	5+3=8	5+6=11	5+9=14
5+1=6	5+4=9	5+7=12	5+10=15
5+2=7	5+5=10	5+8=13	

Как видите, прибавляя любое положительное число к какому-либо числу (в нашем случае к 5), мы всегда получим в ответе число равное или больше 5.

Очень важным моментом является то, что установка нижнего диапазона сложением влияет и на верхний диапазон, сдвигая его вверх. Для понятности приведем практический пример:

Нам необходимо имитировать бросок игрального кубика при помощи ГПСЧ, выдающий числа от 0 до 10. Установим верхнюю границу при помощи деления с остатком. Максимальное значение игрального кубика 6, это означает, что необходимо делить с остатком на 7:

0 mod 7 = 0
1 mod 7 = 1
2 mod 7 = 2

3 mod 7 = 3
4 mod 7 = 4
5 mod 7 = 5

6 mod 7=6
7 mod 7 = 0
8 mod 7 = 1

9 mod 7 = 2
10 mod 7 = 3

Итак, мы получили диапазон от 0 до 6. Но игральный кубик не может выдать 0, поэтому мы должны получившееся число прибавить к 1:

1+(0 mod 7) = 1+0=1
1+ (1 mod 7) = 1+1=2
1+ (2 mod 7) = 1+2=3
1+ (3 mod 7) =1+3= 4
1+ (4 mod 7) =1+4= 5
1+ (5 mod 7) = 1+5=6

1+ (6 mod 7) = 1+6=7
1+ (7 mod 7) =1+0=1
1+ (8 mod 7) = 1+1=2
1+ (9 mod 7) = 1+2=3
1+ (10 mod 7) = 1+3=4

Теперь мы получили диапазон от 1 до 7. Но ведь нам надо до 6! Как исправить данное несоответствие? Нужно делить число с остатком на 7 минус 1 (то число, к которому прибавляем) т.е. m+(x mod (n-m)), где m нижний диапазон. Вот что у нас получится:

1+(0 mod (7-1)) = 1+(0 mod 6)=1
1+ (1 mod (7-1)) = 1+(1 mod 6) =2
1+ (2 mod (7-1)) = 1+(2 mod 6)=3
1+ (3 mod (7-1)) =1+(3 mod 6)=4
1+ (4 mod (7-1)) =1+(4 mod 6)=5
1+ (5 mod (7-1)) = 1+(5 mod 6)=6

1+ (6 mod (7-1)) = 1+(6 mod 6)=1
1+ (7 mod (7-1)) =1+(7 mod 6)=2
1+ (8 mod (7-1)) = 1+(8 mod 6)=3
1+ (9 mod (7-1)) = 1+(9 mod 6)=4
1+ (10 mod (7-1)) = 1+(10 mod 6)=5

Вот полная формула приведения к необходимому диапазону:

$$x\ mod\ (n-m)+m=c$$

Где: x – случайное число, n – делитель , m – самое меньшее число диапазона, c – результат (число из диапазона $m \leq c < n$).

Теперь давайте представим, что у нас ГПСЧ создает числа от 1 до 5, а нам нужно, чтобы псевдослучайные числа были из диапазона от 3 до 4 (т.е. только 3 и 4). Минимальное число (m) в нашем случае будет 3. Т.к. мы получаем числа меньше делителя, то в качестве него мы должны звзять самое ближайшее, но наиболее ближнее число – это цифра 5. Т.е. m=3, n=5. Далее по формуле:

$$x\ mod\ (n-m)+m=x\ mod\ (5-3)+3=x\ mod\ (2)+3$$

Просчитаем:

3+(1 mod 2)=3+1=4
3+(2 mod 2) =3+0=3
3+(3 mod 2)=3+1=4

3+(4 mod 2)=3+0=3
3+(5 mod 2)=3+1=4

Получилось! Мы получили числа из необходимого нам диапазона!

Стоит отметить, что эта формула годна только для положительных чисел. Чтобы привести к нужному диапазону отрицательное число, необходимо при помощи модуля сделать его положительным, привести к нужному диапазону, а затем умножить на -1.

Проблема диапазона итогового сгенерированного числа при помощи XOR

Несмотря на то, что мы используем десятичную систему счисления, не стоит забывать, что компьютер использует двоичную. Это приводит к некоторым трудностям, в том числе и при накапливании случайности при помощи функции XOR.

Все дело в том, что XOR выполняется над числами, которые представлены именно в двоичном виде. Это приводит к тому, что «разброс» получаемых результатов весьма высок. Так, мы можем смешивать числа от 0 до 9, но получить числа гораздо больше 9. Вот наглядный пример «смешивания» чисел 9 и 3:

$$\begin{array}{r} 9 \\ \text{XOR}\ 3 \\ \hline 10 \end{array} \quad \Rightarrow \quad \begin{array}{r} 1\ 0\ 0\ 1 \\ \text{XOR}\ 0\ 0\ 1\ 1 \\ \hline 1\ 0\ 1\ 0 \end{array}$$

$$\begin{array}{r} 9 \\ \text{XOR}\ 6 \\ \hline 15 \end{array} \quad \Rightarrow \quad \begin{array}{r} 1\ 0\ 0\ 1 \\ \text{XOR}\ 0\ 1\ 1\ 0 \\ \hline 1\ 1\ 1\ 1 \end{array}$$

$$\begin{array}{r} 7 \\ \text{XOR}\ 7 \\ \hline 0 \end{array} \quad \Rightarrow \quad \begin{array}{r} 1\ 1\ 1 \\ \text{XOR}\ 1\ 1\ 1 \\ \hline 0\ 0\ 0 \end{array}$$

$$\begin{array}{r} 7 \\ \text{XOR}\ 6 \\ \hline 1 \end{array} \quad \Rightarrow \quad \begin{array}{r} 1\ 1\ 1 \\ \text{XOR}\ 1\ 1\ 0 \\ \hline 0\ 0\ 1 \end{array}$$

Таким образом, может возникнуть ситуация, когда нам нужно привести к необходимому диапазону не только сгенерированные ГПСЧ значения, но и итоговое значение, которое мы получим после проведения всех XOR.

К сожалению, как показывает практика, применение к итоговому значению формулы, описанной выше, приводит к сильному падению случайности. Используя XOR к двум числам, в диапазоне от 0 до 9, мы получаем числа в диапазоне от 0 до 15. Числа 10, 11, 12, 13, 14 и 15 приводятся к диапазону взятием остатка от деления на 10, т.е., по факту, заменяются числами 0, 1, 2, 3, 4, 5. Таким образом, числа от 0 до 5 могут получится не одним, а двумя способами. Это приводит к тому, что этих чисел создается на порядок больше.

Для наглядности приведем пример. Составим таблицу всех возможных значений при «перемешивании» двух чисел от 0 до 9 при помощи XOR:

XOR		Число А									
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Число В	**0**	0	1	2	3	4	5	6	7	8	9
	1	1	0	3	2	5	4	7	6	9	8
	2	2	3	0	1	6	7	4	5	10	11
	3	3	2	1	0	7	6	5	4	11	10
	4	4	5	6	7	0	1	2	3	12	13
	5	5	4	7	6	1	0	3	2	13	12
	6	6	7	4	5	2	3	0	1	14	15
	7	7	6	5	4	3	2	1	0	15	14
	8	8	9	10	11	12	13	14	15	0	1
	9	9	8	11	10	13	12	15	14	1	0

Несмотря на то, что число выбирается случайно, шансы у разных чисел разные, так как в таблице их разное количество. Давайте посчитаем сколько и каких чисел в таблице, а затем рассчитаем случайность.

Число	Количество
0	10
1	10
2	8
3	8
4	8
5	8

Число	Количество
6	8
7	8
8	4
9	4
10	4
11	4

Число	Количество
12	4
13	4
14	4
15	4

Наиболее вероятными являются числа 0 и 1. Их вероятность появления 10/100=0,1. Наименее вероятными являются числа от 8 до 15. Их вероятность появления 4/100=0,04. «Не случайность» равна 0,1-0,04=0,06. Таким образом, случайность генерации чисел 1-0,06=0,94.

Теперь давайте приведем значения таблицы к диапазону от 0 до 9, используя взятие остатка от деления на 10. Мы получим следующую таблицу:

XOR		Число А									
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
ис до	**0**	0	1	2	3	4	5	6	7	8	9

	1	1	0	3	2	5	4	7	6	9	8
	2	2	3	0	1	6	7	4	5	0	1
	3	3	2	1	0	7	6	5	4	1	0
	4	4	5	6	7	0	1	2	3	2	3
	5	5	4	7	6	1	0	3	2	3	2
	6	6	7	4	5	2	3	0	1	4	5
	7	7	6	5	4	3	2	1	0	5	4
	8	8	9	0	1	2	3	4	5	0	1
	9	9	8	1	0	3	2	5	4	1	0

Подсчитаем сколько и каких цифр:

Число	Количество
0	14
1	14
2	12
3	12

Число	Количество
4	12
5	12
6	8
7	8

Число	Количество
8	4
9	4

Наиболее вероятными являются числа 0 и 1: 14/100=0,14. Наименее вероятными являются числа 8 и 9: 4/100=0,04. «Не случайность» в этом случае у нас 0,14-0,04=0,1. Таким образом, получается, что случайность равна 1-0,1=0,9.

0,9 меньше, чем 0,94. Т.е. приведя итоговое сгенерированное число к нужному диапазону, мы уменьшим его случайность.

Чтобы избежать данной проблемы мы должны использовать хитрость: мы будем использовать сгенерированное число только в том случае, если оно соответствует нужному диапазону. Если же нет, то мы генерируем новое число.

Проверяем на практике

В чем суть проверки на практике

Любые идеи, теории и задумки остаются таковыми до тех пор, ока эта задумка не будет воплощена в реальной жизни. Чтобы доказать накопление случайности нам нужно осуществить это на практике, чем мы сейчас и займемся.

В первую очередь опишем сам эксперимент. Суть его такова: ГПСЧ генерирует миллион случайных чисел от 0 до 9. Далее по этим числам вычисляется «не случайность» и случайность ГПСЧ. Почему именно миллион? Потому что это, с точки зрения теории вероятностей, достаточно большое число для проверки накопления случайности. Повторять опыт мы будем по 9 раз, каждый раз увеличивая количество смешиваемых случайных чисел в 2 раза. Действие XOR мы будем проводить 0, 1, 2, 4, 8, 16, 32, 64 и 124 раза.

Механизм накопления случайности будет реализован на 17 ГПСЧ, а соответствующие программы будет написана на 8 языках. Компьютер сам будет генерировать миллион случайных чисел, считать и выдавать уже готовые значения вероятностей.

Общий алгоритм работы программы

Общий алгоритм работы программы прост. Программа, используя ГПСЧ, генерирует случайные числа и, при помощи того же ГПСЧ, накапливает случайность, производя «смешивание» чисел через XOR. После этого, программа посчитает, сколько и каких чисел она создала, а также вычислит вероятности всех 10 чисел (от 0 до 9).

Опишу более подробно.

В начале программы создаются 10 переменных под названием chis0, chis1, chis2, chis3, chis4, chis5, chis6, chis7, chis8, и chis9. Сразу же для этих переменных устанавливается значение 0. Эти переменные нужны для того, чтобы считать сколько и каких чисел выдал генератор. Если ГПСЧ выдал 0, то значение переменной chis0 увеличивается на 1, если 7, то значение переменной chis7. Думаю, суть Вы поняли.

Далее создается переменная-счетчик j, которой сразу присваивается значение 0. Она будет считать количество сгенерированных случайных чисел. Генерация чисел при помощи ГПСЧ будет продолжатся для тех пор, пока j будет меньше миллиона. Это реализуется при помощи цикла while.

Все промежуточные и итоговые значения будут храниться в переменной n. В n будет записано первое случайное число, а затем, при помощи цикла, смешиваться с другим случайным числом:

```
n = random();
for (i = 0; i<"XOR"; i++) {
    n = n XOR random();
}
```

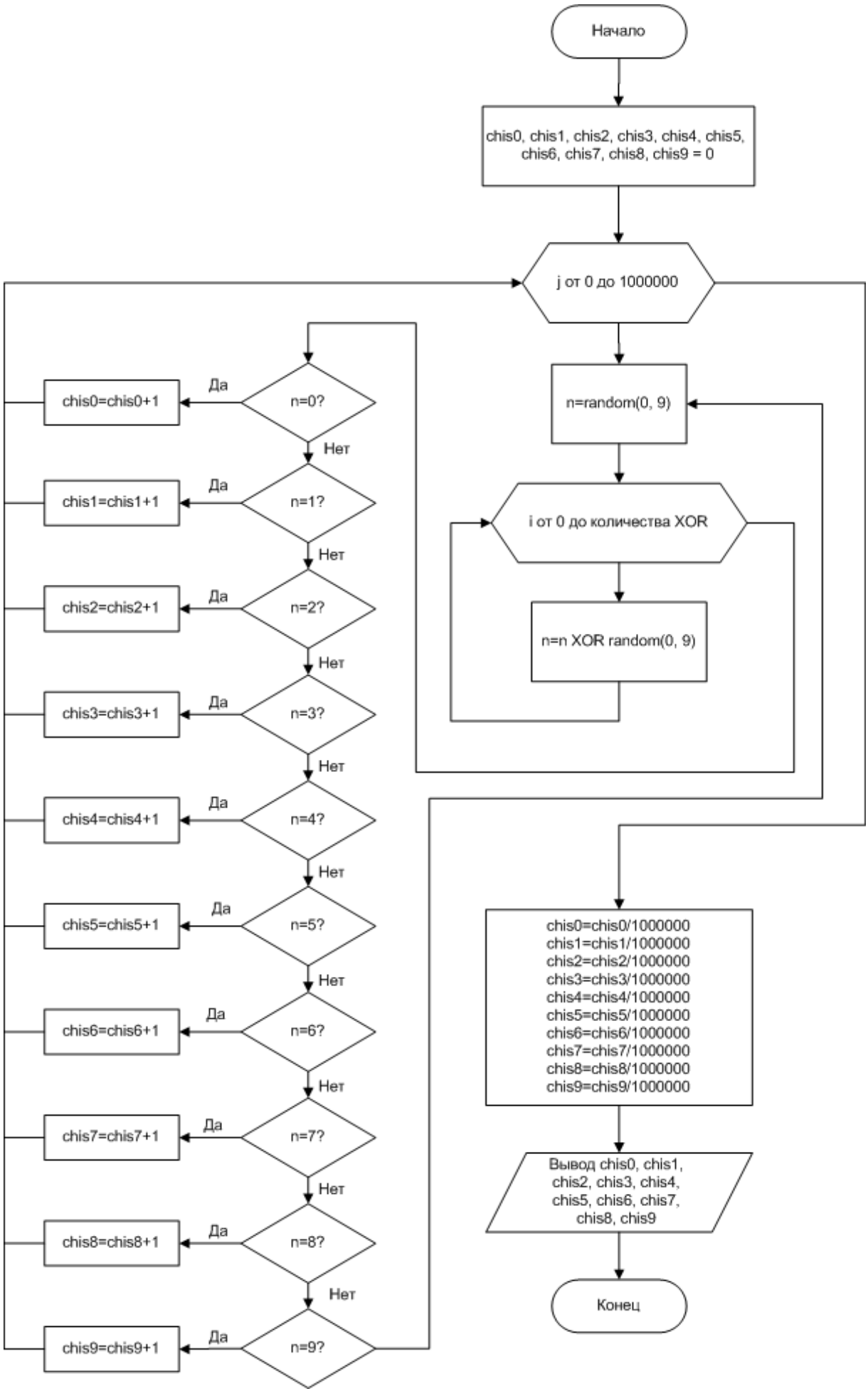
```
n = random()
for i = 0 to "XOR" do
    n = n XOR random ()
Next
```

Вместо «XOR» вставляется количество «смешиваемых чисел». В каждом опыте оно разное и соответствует тем количествам раз, которые мы обговорили выше (0, 1, 2, 4, 8, 16, 32, 64, 124). Получившееся число проверяется при помощи конструкции if на соответствие нашему диапазону. Если что не соответствует, то оно отбрасывается. Если число соответствует, то оно относится к одному из чисел от 0 до 9, чья переменная chis увеличится на 1, после чего счетчик j увеличивается на единицу.

Для того, чтобы подсчитать вероятность появления каждого из чисел от 0 до 9, мы просто делим переменную chis, в которой хранится число, указывающее на то, сколько было сгенерировано таких чисел, на миллион.

Полученные значения вносятся в таблицу, после чего рассчитывается Q, S, ΔS и k.

Для наглядности приведу блок-схему:


Начало
chis0, chis1, chis2, chis3, chis4, chis5, chis6, chis7, chis8, chis9 = 0
j от 0 до 1000000
n=random(0, 9)
i от 0 до количества XOR
n=n XOR random(0, 9)
n=0?
Да
chis0=chis0+1
Нет
n=1?
Да
chis1=chis1+1
Нет
n=2?
Да
chis2=chis2+1
Нет
n=3?
Да
chis3=chis3+1
Нет
n=4?
Да
chis4=chis4+1
Нет
n=5?
Да
chis5=chis5+1
Нет
n=6?
Да
chis6=chis6+1
Нет
n=7?
Да
chis7=chis7+1
Нет
n=8?
Да
chis8=chis8+1
Нет
n=9?
Да
chis9=chis9+1
chis0=chis0/1000000
chis1=chis1/1000000
chis2=chis2/1000000
chis3=chis3/1000000
chis4=chis4/1000000
chis5=chis5/1000000
chis6=chis6/1000000
chis7=chis7/1000000
chis8=chis8/1000000
chis9=chis9/1000000
Вывод chis0, chis1, chis2, chis3, chis4, chis5, chis6, chis7, chis8, chis9
Конец

PHP

Сначала мы проведем эксперименты на языке программирования PHP, версии 8.0.12.

У PHP существует 5 различных ГПСЧ:

1. Вихрь Марсенна – разработан в 1997 году японскими учеными. Его главной особенностью является быстрота работы, однако случайность создаваемых чисел ниже всех остальных ГПСЧ, доступных в PHP. В документации вообще не рекомендован к использованию. Вызывается командой rand.
2. CryptGenRandom – ГПСЧ, входивший в состав 32-х битных версий Windows, создан для использования в криптографии. PHP имеет возможность воспользоваться им для генерации случайных чисел, вызывается она командой random_int(). Стоит отметить, что в документации языка PHP не рекомендуется использовать данный ГПСЧ для криптографии.
3. CNG – программный интерфейс приложений (API), стандартный для Windows, начиная с Vista, имеющий свой собственный ГПСЧ. Рекомендован W3C как ГПСЧ для шифрования информации. PHP имеет возможность использовать его для создания случайных чисел, которая вызывается командой random_bytes().
4. md_rand – стандартный ГПСЧ, встроенный в самую распространенную библиотеку шифрования – OpenSSL. Данная библиотека считается эталонной для реализации шифрования и используется повсеместно. Вызывается данный ГПСЧ в языке программирования PHP командой openssl_random_pseudo_bytes().
5. Генератор уникальных ID – генератор, генерирующий уникальные ID для использования в сессиях и базах данных. Основан на системном времени компьютера, однако использует дополнительные источники энтропии, что делает значения гораздо более случайными, чем если бы мы использовали время. Запускается генератор командой uniqid(“”, true).

Код программы для эксперимента с Вихрем Марсдена:

```
<?php


//создаем переменные для учета чисел

$chis0=0;

$chis1=0;
```

```
$chis2=0;
$chis3=0;
$chis4=0;
$chis5=0;
$chis6=0;
$chis7=0;
$chis8=0;
$chis9=0;

//создаем переменную для подсчета количества
сгенерированных чисел
$j=0;

//запускаем генерацию миллиона случайных чисел
while ($j<1000000) {
    $n=rand(0, 9); //генерируем случайное число

    //накапливаем случайность при помощи XOR
    for ($i=0; $i<”Количество XOR”; $i++) {
        $n=$n^rand(0, 9);
    }

    //Сортируем числа
    switch ($n) {
        //Если сгенерированное число равно 0
        case 0:
```

```
    $chis0=$chis0+1; //Увеличиваем chis0 на 1
    $j=$j+1; //Увеличиваем j на 1
break; //Выходим из case
//Если сгенерированное число равно 1
case 1:
    $chis1=$chis1+1; //Увеличиваем chis1 на 1
    $j=$j+1; //Увеличиваем j на 1
break; //Выходим из case
//Если сгенерированное число равно 2
case 2:
    $chis2=$chis2+1; //Увеличиваем chis2 на 1
    $j=$j+1; //Увеличиваем j на 1
break; //Выходим из case
//Если сгенерированное число равно 3
case 3:
    $chis3=$chis3+1; //Увеличиваем chis3 на 1
    $j=$j+1; //Увеличиваем j на 1
break; //Выходим из case
//Если сгенерированное число равно 4
case 4:
    $chis4=$chis4+1; //Увеличиваем chis4 на 1
    $j=$j+1; //Увеличиваем j на 1
break; //Выходим из case
//Если сгенерированное число равно 5
case 5:
    $chis5=$chis5+1; //Увеличиваем chis5 на 1
```

```
            $j=$j+1; //Увеличиваем j на 1
        break; //Выходим из case
        //Если сгенерированное число равно 6
        case 6:
            $chis6=$chis6+1; //Увеличиваем chis6 на 1
            $j=$j+1; //Увеличиваем j на 1
        break; //Выходим из case
        //Если сгенерированное число равно 7
        case 7:
            $chis7=$chis7+1; //Увеличиваем chis7 на 1
            $j=$j+1; //Увеличиваем j на 1
        break; //Выходим из case
        //Если сгенерированное число равно 8
        case 8:
            $chis8=$chis8+1; //Увеличиваем chis8 на 1
            $j=$j+1; //Увеличиваем j на 1
        break; //Выходим из case
        //Если сгенерированное число равно 9
        case 9:
            $chis9=$chis9+1; //Увеличиваем chis9 на 1
            $j=$j+1; //Увеличиваем j на 1
        break; //Выходим из case
        }
}
```

```
//выводим вероятности появления чисел от 0 до 9
echo $chis0/1000000; //Вероятность числа 0
echo '<br>'; //Отступ для удобочитаемости вывода
echo $chis1/1000000; //Вероятность числа 1
echo '<br>'; //Отступ для удобочитаемости вывода
echo  $chis2/1000000; //Вероятность числа 2
echo '<br>'; //Отступ для удобочитаемости вывода
echo $chis3/1000000; //Вероятность числа 3
echo '<br>'; //Отступ для удобочитаемости вывода
echo $chis4/1000000; //Вероятность числа 4
echo '<br>'; //Отступ для удобочитаемости вывода
echo $chis5/1000000; //Вероятность числа 5
echo '<br>'; //Отступ для удобочитаемости вывода
echo $chis6/1000000; //Вероятность числа 6
echo '<br>'; //Отступ для удобочитаемости вывода
echo $chis7/1000000; //Вероятность числа 7
echo '<br>'; //Отступ для удобочитаемости вывода
echo $chis8/1000000; //Вероятность числа 8
echo '<br>'; //Отступ для удобочитаемости вывода
echo $chis9/1000000; //Вероятность числа 9
?>
```

Результат проведенных экспериментов показан в таблице ниже:

Вихрь Марсенна					
XOR	P_{max}	P_{min}	Q	S	ΔS
0	0,100455	0,099453	0,001002	0,998998	
1	0,10041	0,039894	0,060516	0,939484	–0,05951
2	0,076341	0,051697	0,024644	0,975356	+0,035872
4	0,067647	0,057557	0,01009	0,98991	+0,014554

8	0,063324	0,061842	0,001482	0,998518	+0,008608
16	0,062728	0,062065	0,000663	0,999337	+0,000819
32	0,063065	0,062143	0,000922	0,999078	–0,00026
64	0,062812	0,06217	0,000642	0,999358	+0,00028
128	0,062804	0,061979	0,000825	0,999175	+0,00018

Вывод: В результате увеличения числа смешиваемых случайных чисел случайность генерируемого числа вначале упала в 10 раз, но при дальнейшем увеличении случайность увеличилась во 100 раз. Таким образом, нам удалось увеличить случайность генерируемых чисел в 10 раз от начального значения! Накопление случайности работает. Средней коэффициент накопления k = 0,00000142968.

Теперь проведем тот же опыт, но только с ГПСЧ CryptGenRandom.

Код программы:

```
<?php


//создаем переменные chis

$chis0=0;

$chis1=0;

$chis2=0;

$chis3=0;

$chis4=0;

$chis5=0;

$chis6=0;

$chis7=0;

$chis8=0;

$chis9=0;


//создаем переменную для подсчета количества сгенерированных чисел

$j=0;
```

```
//запускаем генерацию миллиона случайных чисел
while ($j<1000000) {

    //Генерируем случайное число
    $n=random_int(0, 9);

    //накапливаем случайность через XOR
    for ($i=0; $i<”Количество XOR”; $i++) {
        $n=$n^random_int(0, 9);
    }

    //Сортируем числа
    switch ($n) {
        //Если сгенерированное число равно 0
        case 0:
            $chis0=$chis0+1; //Увеличиваем chis0 на 1
            $j=$j+1; //Увеличиваем j на 1
        break; //Выходим из case
        //Если сгенерированное число равно 1
        case 1:
            $chis1=$chis1+1; //Увеличиваем chis1 на 1
            $j=$j+1; //Увеличиваем j на 1
        break; //Выходим из case
        //Если сгенерированное число равно 2
        case 2:
            $chis2=$chis2+1; //Увеличиваем chis2 на 1
            $j=$j+1; //Увеличиваем j на 1
```

```
break; //Выходим из case
//Если сгенерированное число равно 3
case 3:
    $chis3=$chis3+1; //Увеличиваем chis3 на 1
    $j=$j+1; //Увеличиваем j на 1
break; //Выходим из case
//Если сгенерированное число равно 4
case 4:
    $chis4=$chis4+1; //Увеличиваем chis4 на 1
    $j=$j+1; //Увеличиваем j на 1
break; //Выходим из case
//Если сгенерированное число равно 5
case 5:
    $chis5=$chis5+1; //Увеличиваем chis5 на 1
    $j=$j+1; //Увеличиваем j на 1
break; //Выходим из case
//Если сгенерированное число равно 6
case 6:
    $chis6=$chis6+1; //Увеличиваем chis6 на 1
    $j=$j+1; //Увеличиваем j на 1
break; //Выходим из case
//Если сгенерированное число равно 7
case 7:
    $chis7=$chis7+1; //Увеличиваем chis7 на 1
    $j=$j+1; //Увеличиваем j на 1
break; //Выходим из case
//Если сгенерированное число равно 8
case 8:
```

```
            $chis8=$chis8+1; //Увеличиваем chis8 на 1
            $j=$j+1; //Увеличиваем j на 1
        break; //Выходим из case
        //Если сгенерированное число равно 9
        case 9:
            $chis9=$chis9+1; //Увеличиваем chis9 на 1
            $j=$j+1; //Увеличиваем j на 1
        break; //Выходим из case
        }
}


//выводим вероятности появления чисел от 0 до 9
echo $chis0/1000000; //Вероятность числа 0
echo '<br>'; //Отступ для удобочитаемости вывода
echo $chis1/1000000; //Вероятность числа 1
echo '<br>'; //Отступ для удобочитаемости вывода
echo  $chis2/1000000; //Вероятность числа 2
echo '<br>'; //Отступ для удобочитаемости вывода
echo $chis3/1000000; //Вероятность числа 3
echo '<br>'; //Отступ для удобочитаемости вывода
echo $chis4/1000000; //Вероятность числа 4
echo '<br>'; //Отступ для удобочитаемости вывода
echo $chis5/1000000; //Вероятность числа 5
echo '<br>'; //Отступ для удобочитаемости вывода
echo $chis6/1000000; //Вероятность числа 6
echo '<br>'; //Отступ для удобочитаемости вывода
echo $chis7/1000000; //Вероятность числа 7
```

```
echo '<br>'; //Отступ для удобочитаемости вывода
echo $chis8/1000000; //Вероятность числа 8
echo '<br>'; //Отступ для удобочитаемости вывода
echo $chis9/1000000; //Вероятность числа 9
?>
```

Результаты экспериментов представлены в таблице ниже:

CryptGenRandom					
XOR	P_{max}	P_{min}	Q	S	ΔS
0	0.100479	0.099535	0,000944	0,999056	
1	0,100442	0,03983	0,060612	0,939388	–0,059668
2	0,076483	0,052049	0,024434	0,975566	+0,036178
4	0,067914	0,057614	0,0103	0,9897	+0,014134
8	0,063483	0,061473	0,002010	0,99799	+0,00829
16	0,062699	0,062135	0,000564	0,999436	+0,001446
32	0,062959	0,062087	0,000872	0,999128	–0,000308
64	0,062925	0,062337	0,000588	0,999412	+0,000284
128	0,062848	0,062031	0,000817	0,999183	–0,000229

Вывод: В результате увеличения числа смешиваемых случайных чисел случайность генерируемого числа вначале упала в 100 раз, но при дальнейшем увеличении случайность увеличилась во 100 раз. Далее накопление случайности не происходит потому, что мы накопили максимальную случайность! Учитывая, что ГПСЧ CryptGenRandom изначально генерировал случайные числа с такой случайностью, его можно считать идеальным. Коэффициент средней коэффициент накопления k=0,000015875

Следующим у нас на очереди идет ГПСЧ CNG. Данный ГПСЧ вызывается командой random_bytes, в скобках которой обязательно должно быть указано количество генерируемых байт. В нашем случае мы ставим 1:

```
$n=random_bytes(1);
```

Его особенность заключается в том, что он создает не случайные числа, а случайные байты (набор случайных 8 битов), которые компьютер видит как строку (string). Это говорит нам о том, что мы должны перевести строку в числовой формат. Для этого мы должны использовать команду ord(), которая переводит байт string в число integer от 0 до 255:

```
$n=ord(random_bytes(1));
```

В нашем эксперименте требуется генерировать случайные числа в диапазоне от 0 до 9, поэтому нам нужно привести генерируемы числа к данному диапазону взятием остатка от деления на 10:

```
$n=ord(random_bytes(1))%10;
```

Теперь в переменной $n мы имеем случайное число от 0 до 9, созданное при помощи ГПСЧ CNG.
Приступим к нашему эксперименту. Вот полный код программы:

```
<?php

//создаем переменные chis
$chis0=0;
$chis1=0;
$chis2=0;
$chis3=0;
$chis4=0;
$chis5=0;
$chis6=0;
$chis7=0;
$chis8=0;
$chis9=0;

//создаем переменную для подсчета количества
сгенерированных чисел
$j=0;

//запускаем генерацию миллиона случайных чисел
while ($j<1000000) {

    //генерируем случайное число
    $n=ord(random_bytes(1))%10;

    //накапливаем случайность при помощи XOR
    for ($i=0; $i<”Количество XOR”; $i++) {
        $n=$n^(ord(random_bytes(1))%10);
```

```
    }

    //Сортируем числа
    switch ($n) {
        //Если сгенерированное число равно 0
        case 0:
            $chis0=$chis0+1; //Увеличиваем chis0
на 1
            $j=$j+1; //Увеличиваем j на 1
        break; //Выходим из case
        //Если сгенерированное число равно 1
        case 1:
            $chis1=$chis1+1; //Увеличиваем chis1
на 1
            $j=$j+1; //Увеличиваем j на 1
        break; //Выходим из case
        //Если сгенерированное число равно 2
        case 2:
            $chis2=$chis2+1; //Увеличиваем chis2
на 1
            $j=$j+1; //Увеличиваем j на 1
        break; //Выходим из case
        //Если сгенерированное число равно 3
        case 3:
            $chis3=$chis3+1; //Увеличиваем chis3
на 1
            $j=$j+1; //Увеличиваем j на 1
        break; //Выходим из case
        //Если сгенерированное число равно 4
        case 4:
            $chis4=$chis4+1; //Увеличиваем chis4
на 1
            $j=$j+1; //Увеличиваем j на 1
        break; //Выходим из case
        //Если сгенерированное число равно 5
        case 5:
            $chis5=$chis5+1; //Увеличиваем chis5
на 1
            $j=$j+1; //Увеличиваем j на 1
        break; //Выходим из case
```

```
        //Если сгенерированное число равно 6
        case 6:
            $chis6=$chis6+1; //Увеличиваем chis6
на 1
            $j=$j+1; //Увеличиваем j на 1
        break; //Выходим из case
        //Если сгенерированное число равно 7
        case 7:
            $chis7=$chis7+1; //Увеличиваем chis7
на 1
            $j=$j+1; //Увеличиваем j на 1
        break; //Выходим из case
        //Если сгенерированное число равно 8
        case 8:
            $chis8=$chis8+1; //Увеличиваем chis8
на 1
            $j=$j+1; //Увеличиваем j на 1
        break; //Выходим из case
        //Если сгенерированное число равно 9
        case 9:
            $chis9=$chis9+1; //Увеличиваем chis9
на 1
            $j=$j+1; //Увеличиваем j на 1
        break; //Выходим из case
        }
}

//выводим вероятности появления чисел от 0 до 9
echo $chis0/1000000; //Вероятность числа 0
echo '<br>'; //Отступ для удобочитаемости вывода
echo $chis1/1000000; //Вероятность числа 1
echo '<br>'; //Отступ для удобочитаемости вывода
echo  $chis2/1000000; //Вероятность числа 2
echo '<br>'; //Отступ для удобочитаемости вывода
echo $chis3/1000000; //Вероятность числа 3
echo '<br>'; //Отступ для удобочитаемости вывода
echo $chis4/1000000; //Вероятность числа 4
echo '<br>'; //Отступ для удобочитаемости вывода
echo $chis5/1000000; //Вероятность числа 5
echo '<br>'; //Отступ для удобочитаемости вывода
```

```
    echo $chis6/1000000; //Вероятность числа 6
    echo '<br>'; //Отступ для удобочитаемости вывода
    echo $chis7/1000000; //Вероятность числа 7
    echo '<br>'; //Отступ для удобочитаемости вывода
    echo $chis8/1000000; //Вероятность числа 8
    echo '<br>'; //Отступ для удобочитаемости вывода
    echo $chis9/1000000; //Вероятность числа 9
?>
```

Результаты представлены в таблице:

CNG					
XOR	P_{max}	P_{min}	Q	S	ΔS
0	0,101876	0,097402	0,004474	0,995526	
1	0,100359	0,039695	0,060664	0,939336	–0,056190
2	0,077306	0,051089	0,026217	0,973783	+0,034447
4	0,067978	0,05746	0,010518	0,989482	+0,015699
8	0,063539	0,061882	0,001657	0,998343	+0,008861
16	0,062933	0,062211	0,000722	0,999278	+0,000935
32	0,062918	0,06186	0,001058	0,998942	–0,000336
64	0,062903	0,062114	0,000789	0,999211	+0,000269
128	0,062948	0,062257	0,000691	0,999309	+0,000098

Вывод: В результате увеличения числа смешиваемых случайных чисел случайность генерируемого числа в начале уменьшилась в 10 раз, а затем увеличилась в 100 раз! Таким образом, нам удалось увеличить случайность генерируемых чисел в 10 раз от первоначального значения! Средней коэффициент накопления k=0,000472875.

Перейдем к испытанию следующего ГПСЧ, а именно – ГПСЧ, встроенного в OpenSSL. Вызывается данный ГПСЧ командой openssl_random_pseudo_bytes() с обязательным указанием количества генерируемых байт (в нашем случае 1). Как Вы уже могли догадаться, данный ГПСЧ, как и CNG, создает не числа, а байты. В байты приводим к генерации числа от 0 до 9, как и в предыдущем примере, при помощи команды ord и взятия остатка от деления на 10:

```
$n=ord(openssl_random_pseudo_bytes(1))%10;
```

Вот код программы:

```
<?php
```

```
//создаем переменные chis
$chis0=0;
$chis1=0;
$chis2=0;
$chis3=0;
$chis4=0;
$chis5=0;
$chis6=0;
$chis7=0;
$chis8=0;
$chis9=0;

//создаем переменную для подсчета количества
сгенерированных чисел
$j=0;

//запускаем генерацию миллиона случайных чисел
while ($j<1000000) {

    //генерируем случайное число
    $n=ord(openssl_random_pseudo_bytes(1))%10;

    //накапливаем случайность через XOR
    for ($i=0; $i<”Количество XOR”; $i++) {
        $n=$n^(ord(openssl_random_pseudo_bytes(1))%10);
    }
```

```
//Сортируем числа
switch ($n) {
    //Если сгенерированное число равно 0
    case 0:
        $chis0=$chis0+1; //Увеличиваем chis0 на 1
        $j=$j+1; //Увеличиваем j на 1
    break; //Выходим из case
    //Если сгенерированное число равно 1
    case 1:
        $chis1=$chis1+1; //Увеличиваем chis1 на 1
        $j=$j+1; //Увеличиваем j на 1
    break; //Выходим из case
    //Если сгенерированное число равно 2
    case 2:
        $chis2=$chis2+1; //Увеличиваем chis2 на 1
        $j=$j+1; //Увеличиваем j на 1
    break; //Выходим из case
    //Если сгенерированное число равно 3
    case 3:
        $chis3=$chis3+1; //Увеличиваем chis3 на 1
        $j=$j+1; //Увеличиваем j на 1
    break; //Выходим из case
    //Если сгенерированное число равно 4
    case 4:
        $chis4=$chis4+1; //Увеличиваем chis4 на 1
```

```
    $j=$j+1; //Увеличиваем j на 1
break; //Выходим из case
//Если сгенерированное число равно 5
case 5:
    $chis5=$chis5+1; //Увеличиваем chis5 на 1
    $j=$j+1; //Увеличиваем j на 1
break; //Выходим из case
//Если сгенерированное число равно 6
case 6:
    $chis6=$chis6+1; //Увеличиваем chis6 на 1
    $j=$j+1; //Увеличиваем j на 1
break; //Выходим из case
//Если сгенерированное число равно 7
case 7:
    $chis7=$chis7+1; //Увеличиваем chis7 на 1
    $j=$j+1; //Увеличиваем j на 1
break; //Выходим из case
//Если сгенерированное число равно 8
case 8:
    $chis8=$chis8+1; //Увеличиваем chis8 на 1
    $j=$j+1; //Увеличиваем j на 1
break; //Выходим из case
//Если сгенерированное число равно 9
case 9:
    $chis9=$chis9+1; //Увеличиваем chis9 на 1
    $j=$j+1; //Увеличиваем j на 1
```

```
        break; //Выходим из case
        }
}


//выводим вероятности появления чисел от 0 до 9
echo $chis0/1000000; //Вероятность числа 0
echo '<br>'; //Отступ для удобочитаемости вывода
echo $chis1/1000000; //Вероятность числа 1
echo '<br>'; //Отступ для удобочитаемости вывода
echo  $chis2/1000000; //Вероятность числа 2
echo '<br>'; //Отступ для удобочитаемости вывода
echo $chis3/1000000; //Вероятность числа 3
echo '<br>'; //Отступ для удобочитаемости вывода
echo $chis4/1000000; //Вероятность числа 4
echo '<br>'; //Отступ для удобочитаемости вывода
echo $chis5/1000000; //Вероятность числа 5
echo '<br>'; //Отступ для удобочитаемости вывода
echo $chis6/1000000; //Вероятность числа 6
echo '<br>'; //Отступ для удобочитаемости вывода
echo $chis7/1000000; //Вероятность числа 7
echo '<br>'; //Отступ для удобочитаемости вывода
echo $chis8/1000000; //Вероятность числа 8
echo '<br>'; //Отступ для удобочитаемости вывода
echo $chis9/1000000; //Вероятность числа 9
?>
```

Вот таблица результатов:

md_rand					
XOR	P_{max}	P_{min}	Q	S	ΔS
0	0,101812	0,097644	0,004168	0,995832	
1	0,100062	0,039353	0,060709	0,939291	– 0,056541
2	0,077049	0,050984	0,026065	0,973935	+0,034644
4	0,067964	0,057229	0,010735	0,989265	+0,015330
8	0,063515	0,061758	0,001757	0,998243	+0,008978
16	0,062879	0,062097	0,000782	0,999218	+0,000975
32	0,062883	0,062219	0,000664	0,999336	+0,000118
64	0,062692	0,062009	0,000683	0,999317	– 0,000019
128	0,063059	0,062066	0,000993	0,999007	– 0,000310

Вывод: В результате увеличения числа смешиваемых случайных чисел случайность генерируемого числа в начале уменьшилась в 10 раз, а затем увеличилась в 100 раз! Таким образом, нам удалось увеличить случайность генерируемых чисел в 10 раз от первоначального значения! Средний коэффициент накопления k=0,000396875.
Давайте перейдем к последнему ГПСЧ - функции генерации ID.
Генератор ID генерирует случайное число на основе времени, установленном на компьютере. Данный ID представляет собой строку из символов и чисел. При этом, чаще всего изменяется последний символ строки, который всегда является цифрой. Именно поэтому мы и будем брать его, обрезая строку командой substr.

Вот код программы:

```
<?php

//создаем переменные chis
$chis0=0;
$chis1=0;
$chis2=0;
$chis3=0;
$chis4=0;
$chis5=0;
$chis6=0;
$chis7=0;
$chis8=0;
$chis9=0;
```

```
//создаем переменную для подсчета количества
сгенерированных чисел
$j=0;

//запускаем генерацию миллиона случайных чисел
while ($j<1000000) {

    //генерируем случайное число
    $n=substr(uniqid("", true), -1, 1);

    //накапливаем случайность при помощи XOR
    for ($i=0; $i<”Нужное количество XOR”; $i++) {
        $n=$n^(substr(uniqid("", true), -1, 1));
    }

    //Сортируем числа
    switch ($n) {
        //Если сгенерированное число равно 0
        case 0:
            $chis0=$chis0+1; //Увеличиваем chis0
на 1
            $j=$j+1; //Увеличиваем j на 1
        break; //Выходим из case
        //Если сгенерированное число равно 1
        case 1:
            $chis1=$chis1+1; //Увеличиваем chis1
на 1
            $j=$j+1; //Увеличиваем j на 1
        break; //Выходим из case
        //Если сгенерированное число равно 2
        case 2:
            $chis2=$chis2+1; //Увеличиваем chis2
на 1
            $j=$j+1; //Увеличиваем j на 1
        break; //Выходим из case
        //Если сгенерированное число равно 3
        case 3:
            $chis3=$chis3+1; //Увеличиваем chis3
на 1
            $j=$j+1; //Увеличиваем j на 1
```

```
        break; //Выходим из case
        //Если сгенерированное число равно 4
        case 4:
            $chis4=$chis4+1; //Увеличиваем chis4
на 1
            $j=$j+1; //Увеличиваем j на 1
        break; //Выходим из case
        //Если сгенерированное число равно 5
        case 5:
            $chis5=$chis5+1; //Увеличиваем chis5
на 1
            $j=$j+1; //Увеличиваем j на 1
        break; //Выходим из case
        //Если сгенерированное число равно 6
        case 6:
            $chis6=$chis6+1; //Увеличиваем chis6
на 1
            $j=$j+1; //Увеличиваем j на 1
        break; //Выходим из case
        //Если сгенерированное число равно 7
        case 7:
            $chis7=$chis7+1; //Увеличиваем chis7
на 1
            $j=$j+1; //Увеличиваем j на 1
        break; //Выходим из case
        //Если сгенерированное число равно 8
        case 8:
            $chis8=$chis8+1; //Увеличиваем chis8
на 1
            $j=$j+1; //Увеличиваем j на 1
        break; //Выходим из case
        //Если сгенерированное число равно 9
        case 9:
            $chis9=$chis9+1; //Увеличиваем chis9
на 1
            $j=$j+1; //Увеличиваем j на 1
        break; //Выходим из case
        }
}
```

```
    //выводим вероятности появления чисел от 0 до 9
    echo $chis0/1000000; //Вероятность числа 0
    echo '<br>'; //Отступ для удобочитаемости вывода
    echo $chis1/1000000; //Вероятность числа 1
    echo '<br>'; //Отступ для удобочитаемости вывода
    echo  $chis2/1000000; //Вероятность числа 2
    echo '<br>'; //Отступ для удобочитаемости вывода
    echo $chis3/1000000; //Вероятность числа 3
    echo '<br>'; //Отступ для удобочитаемости вывода
    echo $chis4/1000000; //Вероятность числа 4
    echo '<br>'; //Отступ для удобочитаемости вывода
    echo $chis5/1000000; //Вероятность числа 5
    echo '<br>'; //Отступ для удобочитаемости вывода
    echo $chis6/1000000; //Вероятность числа 6
    echo '<br>'; //Отступ для удобочитаемости вывода
    echo $chis7/1000000; //Вероятность числа 7
    echo '<br>'; //Отступ для удобочитаемости вывода
    echo $chis8/1000000; //Вероятность числа 8
    echo '<br>'; //Отступ для удобочитаемости вывода
    echo $chis9/1000000; //Вероятность числа 9
?>
```

Результат выполнения программы приведен в таблице:

Генератор ID					
XOR	P_{max}	P_{min}	Q	S	ΔS
0	0,100424	0,099477	0,000947	0,999053	
1	0	0	1	0	– 0,999053
2	0,076208	0,051963	0,024245	0,975755	+0,975755
4	0,067571	0,057473	0,010098	0,989902	+0,014147
8	0,063514	0,062029	0,001485	0,998515	+0,008613
16	0,062822	0,062256	0,000566	0,999434	+0,000919
32	0,062867	0,061896	0,000971	0,999029	– 0,000405
64	0,06267	0,061791	0,000879	0,999121	+0,000092
128	0,063378	0,062129	0,001249	0,998751	– 0,000370

Вывод: В результате увеличения числа смешиваемых случайных чисел, в начале, случайность полностью пропала, но потом возросла до максимально-возможного уровня. Учитывая, что начальная и конечная случайность одинаковые. Накопление случайности работает! Стоит отметить, что данная функция изначально выдавала числа с максимальной случайностью. Средний коэффициент накопления k=−0,00003775

C

С языком программирования PHP мы разобрались, теперь давайте перейдем к C. На практике мы воспользуемся стандартом ANSI (ANSI C), т.к. данный стандарт является универсальным и запускается на большинстве компиляторов C.

Язык программирования C имеет наиболее распространенный и простой ГПСЧ, который обладает очень плохой случайностью. Его суть такова: берется какое-либо число (которое должен ввести пользователь), умножается на большое число (чаще всего 1103515245), затем складывается с другим большим числом (чаще всего12345). Получившееся число делят на 65536, а затем берут остаток от деления числа на еще одно большое число (чаще всего 32768).

$$y = \frac{x \cdot 1103515245 + 12345}{65536} \%32768$$

Считается, что если число умножить на большое число, а затем взять остаток от деления, то мы получим достаточно случайное число. К сожалению, это не так, и такая случайность не является достаточной для криптографии. Кроме того, если пользователь каждый раз будет вводить одно и то же число, он получит один и тот-же результат. Чтобы получать разные псевдослучайные числа, нужно вводить разные числа.

Чтобы пользователю не приходилось вводить числа в ручную, для этого существует функция srand(). Что бы данная функция каждый раз вводила новое число, можно использовать время, указанное на компьютере. Для этого используется функция time().

Для того, чтобы использовать ГПСЧ C, необходимо подключить библиотеки «stdlib.h» и «time.h»:

```
#include <stdlib.h>

#include <time.h>
```

Генерация случайного числа производится двумя командами: выставление нового числа функцией time(NULL) и, непосредственно, генерирование случайного числа с сохранением его в целочисленную переменную (integer) n:

```
srand(time(NULL));

n=rand();
```

Учитывая, что ГПСЧ языка С генерирует случайные числа в диапазоне от 0 до 32767, нам необходимо привести его к диапазону от 0 до 9. Для этого необходимо воспользоваться взятием остатка от деления на 10, которое в языке программирования С обозначается знаком процента «%»:

```
srand(time(NULL));

n=rand()%10;
```

Прежде чем писать саму программу, необходимо пояснить ряд вещей. Учитывая, что мы пишем программу на языке С, а переменные chis при выводе будут иметь дробное значение, их необходимо объявить как переменные типа float. Кроме того, при выводе значений через функцию printf, будет необходимо директиве указать что мы выводим дробные числа:

```
printf("%f\n", "Имя переменной");
```

Вот, собственно, и сам код программы:

```
//Подключаем библиотеки

#include <stdio.h>

#include <stdlib.h>

#include <time.h>


//Сама программа

int main()

{

    //создаем переменные chis

    float chis0=0;

    float chis1=0;

    float chis2=0;

    float chis3=0;

    float chis4=0;

    float chis5=0;
```

```
    float chis6=0;
    float chis7=0;
    float chis8=0;
    float chis9=0;

    //создаем переменную n для значений
    int n=0;

    //создаем переменную для подсчета количества
сгенерированных чисел
    int j=0;

    //запускаем генерацию миллиона случайных чисел
    while (j<1000000) {

        //Генерируем случайное число
        srand(time(NULL));
        n=(rand()%10);

        //накапливаем случайность при помощи XOR
        for (int i=0; i<"Количество XOR"; i++) {
            srand(time(NULL));
            n=n & rand()%10;
        }

        //Сортируем числа
```

```
switch (n) {
     //Если сгенерированное число равно 0
    case 0:
        chis0=chis0+1; //Увеличиваем chis0 на 1
        j=j+1; //Увеличиваем j на 1
        break; //Выходим из case
     //Если сгенерированное число равно 1
    case 1:
        chis1=chis1+1; //Увеличиваем chis1 на 1
        j=j+1; //Увеличиваем j на 1
        break; //Выходим из case
     //Если сгенерированное число равно 2
    case 2:
        chis2=chis2+1; //Увеличиваем chis2 на 1
        j=j+1; //Увеличиваем j на 1
        break; //Выходим из case
     //Если сгенерированное число равно 3
    case 3:
        chis3=chis3+1; //Увеличиваем chis3 на 1
        j=j+1; //Увеличиваем j на 1
        break; //Выходим из case
     //Если сгенерированное число равно 4
    case 4:
        chis4=chis4+1; //Увеличиваем chis4 на 1
        j=j+1; //Увеличиваем j на 1
        break; //Выходим из case
```

```
            //Если сгенерированное число равно 5
            case 5:
                chis5=chis5+1; //Увеличиваем chis5 на 1
                j=j+1; //Увеличиваем j на 1
                break; //Выходим из case
            //Если сгенерированное число равно 6
            case 6:
                chis6=chis6+1; //Увеличиваем chis6 на 1
                j=j+1; //Увеличиваем j на 1
                break; //Выходим из case
            //Если сгенерированное число равно 7
            case 7:
                chis7=chis7+1; //Увеличиваем chis7 на 1
                j=j+1; //Увеличиваем j на 1
                break; //Выходим из case
            //Если сгенерированное число равно 8
            case 8:
                chis8=chis8+1; //Увеличиваем chis8 на 1
                j=j+1; //Увеличиваем j на 1
                break; //Выходим из case
            //Если сгенерированное число равно 9
            case 9:
                chis9=chis9+1; //Увеличиваем chis9 на 1
                j=j+1; //Увеличиваем j на 1
                break; //Выходим из case
        }
```

```
    }

    //выводим вероятности появления чисел от 0 до 9
    printf("%f\n", chis0/1000000);
    printf("%f\n", chis1/1000000);
    printf("%f\n", chis2/1000000);
    printf("%f\n", chis3/1000000);
    printf("%f\n", chis4/1000000);
    printf("%f\n", chis5/1000000);
    printf("%f\n", chis6/1000000);
    printf("%f\n", chis7/1000000);
    printf("%f\n", chis8/1000000);
    printf("%f\n", chis9/1000000);
}
```

Результаты экспериментов сведены в таблицу:

ANSI C					
XOR	P_{max}	P_{min}	Q	S	ΔS
0	0.503015	0	0.503015	0,496985	
1	0.341863	0	0.341863	0,658137	0,161152
2	0.510952	0	0.510952	0,489048	–0,169089
4	0.291768	0	0.291768	0,708232	0,219184
8	0.235149	0	0.235149	0,764851	0,056619
16	0,261613	0,030325	0,231288	0,768712	0,003861
32	0,135102	0,079792	0,05531	0,94469	0,175978
64	0,144033	0,069805	0,074228	0,925772	–0,018918
128	0,131196	0,082256	0,04894	0,95106	0,025288

Вывод: В результате увеличения числа смешиваемых случайных чисел мы смогли увеличить случайность генерируемых псевдослучайных чисел в 10 раз. Возможно и дальнейшее увеличение, однако оно требует смешивания большего числа случайных чисел, нежели 128, поэтому в статье дальнейшее

увеличение мы не рассматриваем. Средней коэффициент накопления k=0,056759375.

C++

С языком С разобрались, теперь перейдем к его ближайшему родственнику: языку программирования С++. Вообще, С++ создавался максимально совместимым с языком С, поэтому он унаследовал от своего прародителя очень многое, в том числе и его ГПСЧ.

Для испытания мы воспользуемся языком программирования С++ стандарта 20 (С++ 20) т.к. он является самой новой версией С++.

ГПСЧ С++ управляется и используется точно также, как и в С: функциями rand() и srand():

```
srand(time(0));
```

```
n=rand();
```

Единственным отличием является то, что ГПСЧ, у С++, входит в стандартную библиотеку, поэтому нам понадобится подключить только одну дополнительную библиотеку для использования функции time. Она называется «ctime»:

```
#include <ctime>
```

Прежде чем перенести программу написанную на С в компилятор С++, необходимо пояснить некоторые различия. Во первых, библиотеки подключаются без окончания «.h». Во вторых, С++ объектно-ориентированный язык, который требует подключения стандартных функций ввода-вывода:

```
using namespace std;
```

Кроме того, вывод осуществляется не при помощи библиотечной функции printf, а при помощи стандартной функции cout:

```
cout << переменная << endl;
```

Кроме этого, в программе ничего менять не надо.

Вот полный код программы для языка С++:

```
//Подключаем библиотеки
```

```
#include <iostream>
```

```
#include <ctime>

//подключаем систему ввода/вывода
using namespace std;

//сама программа
int main()
{
    //создаем переменные chis
    float chis0=0;
    float chis1=0;
    float chis2=0;
    float chis3=0;
    float chis4=0;
    float chis5=0;
    float chis6=0;
    float chis7=0;
    float chis8=0;
    float chis9=0;

    //создаем переменную n для значений
    int n=0;

    //создаем переменную для подсчета количества сгенерированных чисел
    int j=0;
```

```
//запускаем генерацию миллиона случайных чисел
while (j<1000000) {

    //Генерируем случайное число
    srand(time(0));
    n=(rand()%10);

    //накапливаем случайность при помощи XOR
    for (int i=0; i<”Нужное количество XOR”; i++) {
        srand(time(0));
        n=n & rand()%10;
    }

    //Сортируем числа
    switch (n) {
         //Если сгенерированное число равно 0
        case 0:
            chis0=chis0+1; //Увеличиваем chis0 на 1
            j=j+1; //Увеличиваем j на 1
            break; //Выходим из case
         //Если сгенерированное число равно 1
        case 1:
            chis1=chis1+1; //Увеличиваем chis1 на 1
            j=j+1; //Увеличиваем j на 1
            break; //Выходим из case
```

```
 //Если сгенерированное число равно 2
case 2:
    chis2=chis2+1; //Увеличиваем chis2 на 1
    j=j+1; //Увеличиваем j на 1
    break; //Выходим из case
 //Если сгенерированное число равно 3
case 3:
    chis3=chis3+1; //Увеличиваем chis3 на 1
    j=j+1; //Увеличиваем j на 1
    break; //Выходим из case
 //Если сгенерированное число равно 4
case 4:
    chis4=chis4+1; //Увеличиваем chis4 на 1
    j=j+1; //Увеличиваем j на 1
    break; //Выходим из case
 //Если сгенерированное число равно 5
case 5:
    chis5=chis5+1; //Увеличиваем chis5 на 1
    j=j+1; //Увеличиваем j на 1
    break; //Выходим из case
 //Если сгенерированное число равно 6
case 6:
    chis6=chis6+1; //Увеличиваем chis6 на 1
    j=j+1; //Увеличиваем j на 1
    break; //Выходим из case
 //Если сгенерированное число равно 7
```

```
            case 7:
                chis7=chis7+1; //Увеличиваем chis7 на 1
                j=j+1; //Увеличиваем j на 1
                break; //Выходим из case
             //Если сгенерированное число равно 8
            case 8:
                chis8=chis8+1; //Увеличиваем chis8 на 1
                j=j+1; //Увеличиваем j на 1
                break; //Выходим из case
             //Если сгенерированное число равно 9
            case 9:
                chis9=chis9+1; //Увеличиваем chis9 на 1
                j=j+1; //Увеличиваем j на 1
                break; //Выходим из case
        }

}

//выводим вероятности появления чисел от 0 до 9
cout << chis0/1000000 << endl;
cout << chis1/1000000 << endl;
cout << chis2/1000000 << endl;
cout << chis3/1000000 << endl;
cout << chis4/1000000 << endl;
cout << chis5/1000000 << endl;
cout << chis6/1000000 << endl;
```

```
    cout << chis7/1000000 << endl;
    cout << chis8/1000000 << endl;
    cout << chis9/1000000 << endl;
}
```

Результаты экспериментов сведены в таблицу:

C++					
XOR	P_{max}	P_{min}	Q	S	ΔS
0	0,730177	0	0,730177	0,269823	
1	0,493731	0	0,493731	0,506269	+0,236446
2	0,347798	0	0,347798	0,652202	+0,145933
4	0,299978	0	0,299978	0,700022	+0,047820
8	0,250719	0	0,250719	0,749281	+0,049259
16	0,220605	0	0,220605	0,779395	+0,030114
32	0,234883	0,03537	0,199513	0,800487	+0,021092
64	0,127602	0,056287	0,071315	0,928685	+0,128198
128	0,127385	0,075272	0,052113	0,947887	+0,019202

Вывод: В результате увеличения числа смешиваемых случайных чисел мы смогли увеличить случайность генерируемых псевдослучайных чисел в 10 раз. Возможно и дальнейшее увеличение, однако оно требует смешивания большего числа случайных чисел, нежели 128, поэтому в статье дальнейшее увеличение мы не рассматриваем. Средней коэффициент накопления k=0,084758.

C#

Для практических исследований мы будем использовать язык C# стандарта ISO.

C#, как и C++, произошел от языка C. Для обеспечения максимальной совместимости C# имеет точно такой же ГПСЧ, как C и C++. Тем не менее, разработчкики языка C# обратили внимание на ситуацию с плохим ГПСЧ и решили ее, внедрив в C# возможность генерировать псевдослучайные числа при помощи билиотеки CSP, которая является частью программого интерфейса приложений (API), стандартный для Windows, начиная с Vista. Иными словами, аналог CNG для C#.

Кроме того, реализация управления данными ГПСЧ была упрощена – теперь не нужно вводить при помощи функции time новое число, это происходит автоматически.

Стандартный, унаследованный от С ГПСЧ управляется при помощи команды rnd.Next(), где в скобках указываются минимальные и максимальные значения генерируемых случайных чисел.

```
n=rnd.Next(0, 9);
```

Перед использованием ГПСЧ, необходимо создать класс Random.

```
Random rnd = new Random();
```

Полный код программы представлен ниже:

```
//Запускаем директиву using
using System;


//Сама программа
class Program {
    static void Main(string[] args) {

        //Создаем класс Random (ГПСЧ)
        Random rnd = new Random();

        //создаем переменные chis
        float chis0=0;
        float chis1=0;
        float chis2=0;
        float chis3=0;
        float chis4=0;
        float chis5=0;
        float chis6=0;
        float chis7=0;
```

```
        float chis8=0;
        float chis9=0;

        //создаем переменную n для значений
        int n=0;

        //создаем переменную для подсчета количества
сгенерированных чисел
        int j=0;

        //запускаем генерацию миллиона случайных чисел
        while (j<1000000) {

            //Генерируем случайное число
            n=rnd.Next(0, 9);

            //накапливаем случайность при помощи XOR
            for (int i=0; i<”Количество XOR”; i++) {
                n=n^rnd.Next(0, 9);
            }

            //Сортируем числа
            switch (n) {
                //Если сгенерированное число равно 0
                case 0:
```

```
                chis0=chis0+1; //Увеличиваем chis0
на 1
                j=j+1; //Увеличиваем j на 1
                break; //Выходим из case
            //Если сгенерированное число равно 1
            case 1:
                chis1=chis1+1; //Увеличиваем chis1
на 1
                j=j+1; //Увеличиваем j на 1
                break; //Выходим из case
            //Если сгенерированное число равно 2
            case 2:
                chis2=chis2+1; //Увеличиваем chis2
на 1
                j=j+1; //Увеличиваем j на 1
                break; //Выходим из case
            //Если сгенерированное число равно 3
            case 3:
                chis3=chis3+1; //Увеличиваем chis3
на 1
                j=j+1; //Увеличиваем j на 1
                break; //Выходим из case
            //Если сгенерированное число равно 4
            case 4:
                chis4=chis4+1; //Увеличиваем chis4
на 1
                j=j+1; //Увеличиваем j на 1
                break; //Выходим из case
```

```
                //Если сгенерированное число равно 5
                case 5:
                    chis5=chis5+1; //Увеличиваем chis5
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 6
                case 6:
                    chis6=chis6+1; //Увеличиваем chis6
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 7
                case 7:
                    chis7=chis7+1; //Увеличиваем chis7
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 8
                case 8:
                    chis8=chis8+1; //Увеличиваем chis8
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 9
                case 9:
                    chis9=chis9+1; //Увеличиваем chis9
на 1
```

```
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                }
            }

        //выводим вероятности появления чисел от 0 до 9
        Console.WriteLine(chis0/1000000);
        Console.WriteLine(chis1/1000000);
        Console.WriteLine(chis2/1000000);
        Console.WriteLine(chis3/1000000);
        Console.WriteLine(chis4/1000000);
        Console.WriteLine(chis5/1000000);
        Console.WriteLine(chis6/1000000);
        Console.WriteLine(chis7/1000000);
        Console.WriteLine(chis8/1000000);
        Console.WriteLine(chis9/1000000);

        }
}
```

Результаты экспериментов представлены в таблице:

ISO C#					
XOR	P_{max}	P_{min}	Q	S	ΔS
0	0,111531	0	0,111531	0,888469	
1	0,130741	0,028741	0,102	0,898	+0,009531
2	0,115115	0,041147	0,073968	0,926032	+0,028032
4	0,115115	0,041147	0,073968	0,926032	+0,000000
8	0,10432	0,084094	0,020226	0,979774	+0,053742
16	0,100967	0,097449	0,003518	0,996482	+0,016708
32	0,100519	0,099379	0,00114	0,99886	+0,002378

64	0,100576	0,099614	0,000962	0,999038	+0,000178
128	0,100453	0,099317	0,001136	0,998864	–0,000174

Вывод: В результате увеличения числа смешиваемых случайных чисел мы смогли увеличить случайность генерируемых псевдослучайных чисел. Возможно и дальнейшее увеличение, однако оно требует смешивания большего числа случайных чисел, нежели 128, поэтому в статье дальнейшее увеличение мы не рассматриваем. Средней коэффициент накопления k=0,013799375.

Перейдем к испытанию второго ГПСЧ – RNGCryptoServiceProvider. Для того, чтобы воспользоваться данным ГПСЧ, мы должны подключить стандартную криптографическую службу ОС к нашей программе. Это делается через using:

```
using System.Security.Cryptography;
```

Учитывая, что C# является объектно-ориентированным языком программирования, необходимо создать класс ГПСЧ и создать конкретный ГПСЧ с именем rg:

```
private static RNGCryptoServiceProvider rg = new
RNGCryptoServiceProvider();
```

Данный ГПСЧ не создает случайные числа, он заполняет все ячейки указанного массива случайными байтами. Массив, при этом, должен быть типа byte:

```
byte[] rno = new byte[4];
rg.GetBytes(rno);
```

После заполнения массива, мы преобразуем его в случайное число типа integer. Учитывая, что integer имеет размер 32 бита (4 байта), заполняемый массив должен быть размером 4.

Чтобы конвертировать нам массив в случайное число необходимо использовать команду BitConverter.ToInt32(), в скобках которого нужно указать имя массива и с какой по номеру ячейки нужно начать формировать число. В нашем случае, это массив rno, а номер ячейки 0.

```
n=BitConverter.ToInt32(rno, 0);
```

А

В результате выполнения данной строчки мы преобразуем наш массив с 4 случайными байтами в число со знаком в диапазоне от -2147483648 до 2 147 483 647. Нам отрицательные числа не нужны, поэтому получившееся случайное число нам нужно брать в модуле. Модуль в C# вызывается командой Match.Abs():

```
n=Math.Abs(BitConverter.ToInt32(rno, 0));
```

Теперь мы генерируем случайное число от 0 до 2147483648. Данное число нужно привести к нужному нам диапазону по формуле:

$$x\ mod\ (n - m + 1) + m$$

У нас максимальное число 9, а минимальное 0. Таким образом у нас получается:

$$x\ mod\ (9 - 0 + 1) + 0 = x\ mod\ 10$$

В языке программирования C# взятие остатка от деления указывается знаком %, поэтому генерировать случайное число и сохранять в переменную n мы будем так:

```
n=(Math.Abs(BitConverter.ToInt32(rno, 0)))%10;
```

Ниже представлен полностью код всей программы:

```
//Запускаем директиву using
using System;

//Подключаем стандартные криптографические службы
using System.Security.Cryptography;

//Сама программа
class Program {
```

```
    //Создаем класс ГПСЧ
    private static RNGCryptoServiceProvider rg = new
RNGCryptoServiceProvider();

    static void Main(string[] args) {

        //Создаем массив из 4 байтов
        byte[] rno = new byte[4];

        //создаем переменные chis
        float chis0=0;
        float chis1=0;
        float chis2=0;
        float chis3=0;
        float chis4=0;
        float chis5=0;
        float chis6=0;
        float chis7=0;
        float chis8=0;
        float chis9=0;

        //создаем переменную n для значений
        int n=0;

        //создаем переменную для подсчета количества
сгенерированных чисел
```

```
        int j=0;

        //запускаем генерацию миллиона случайных чисел
        while (j<1000000) {

            //Генерируем случайное число
            rg.GetBytes(rno);
            n=(Math.Abs(BitConverter.ToInt32(rno,
0)))%10;

            //накапливаем случайность при помощи XOR
            for (int i=0; i<1; i++) {
                rg.GetBytes(rno);

    n=n^(Math.Abs(BitConverter.ToInt32(rno, 0)))%10;
            }

        //Сортируем числа
        switch (n) {
            //Если сгенерированное число равно 0
           case 0:
               chis0=chis0+1; //Увеличиваем chis0 на 1
               j=j+1; //Увеличиваем j на 1
               break; //Выходим из case
            //Если сгенерированное число равно 1
           case 1:
```

```
            chis1=chis1+1; //Увеличиваем chis1 на 1
            j=j+1; //Увеличиваем j на 1
            break; //Выходим из case
         //Если сгенерированное число равно 2
        case 2:
            chis2=chis2+1; //Увеличиваем chis2 на 1
            j=j+1; //Увеличиваем j на 1
            break; //Выходим из case
         //Если сгенерированное число равно 3
        case 3:
            chis3=chis3+1; //Увеличиваем chis3 на 1
            j=j+1; //Увеличиваем j на 1
            break; //Выходим из case
         //Если сгенерированное число равно 4
        case 4:
            chis4=chis4+1; //Увеличиваем chis4 на 1
            j=j+1; //Увеличиваем j на 1
            break; //Выходим из case
         //Если сгенерированное число равно 5
        case 5:
            chis5=chis5+1; //Увеличиваем chis5 на 1
            j=j+1; //Увеличиваем j на 1
            break; //Выходим из case
         //Если сгенерированное число равно 6
        case 6:
            chis6=chis6+1; //Увеличиваем chis6 на 1
```

```
                j=j+1; //Увеличиваем j на 1
                break; //Выходим из case
            //Если сгенерированное число равно 7
            case 7:
                chis7=chis7+1; //Увеличиваем chis7 на 1
                j=j+1; //Увеличиваем j на 1
                break; //Выходим из case
            //Если сгенерированное число равно 8
            case 8:
                chis8=chis8+1; //Увеличиваем chis8 на 1
                j=j+1; //Увеличиваем j на 1
                break; //Выходим из case
            //Если сгенерированное число равно 9
            case 9:
                chis9=chis9+1; //Увеличиваем chis9 на 1
                j=j+1; //Увеличиваем j на 1
                break; //Выходим из case
        }
    }

    //выводим вероятности появления чисел от 0 до 9
    Console.WriteLine(chis0/1000000);
    Console.WriteLine(chis1/1000000);
    Console.WriteLine(chis2/1000000);
    Console.WriteLine(chis3/1000000);
    Console.WriteLine(chis4/1000000);
```

```
        Console.WriteLine(chis5/1000000);
        Console.WriteLine(chis6/1000000);
        Console.WriteLine(chis7/1000000);
        Console.WriteLine(chis8/1000000);
        Console.WriteLine(chis9/1000000);
        }
}
```

RNGCryptoServiceProvider					
XOR	P_{max}	P_{min}	Q	S	ΔS
0	0,100444	0,09909	0,001354	0,998646	
1	0,131739	0,052674	0,079065	0,920935	–0,077711
2	0,107455	0,072483	0,034972	0,965028	+0,044093
4	0,103452	0,087814	0,015638	0,984362	+0,019334
8	0,100788	0,098037	0,002751	0,997249	+0,012887
16	0,10065	0,099584	0,001066	0,998934	+0,001685
32	0,100313	0,099555	0,000758	0,999242	+0,000308
64	0,100422	0,099698	0,000724	0,999276	+0,000034
128	0,100511	0,099544	0,000967	0,999033	–0,000243

Вывод: В результате увеличения числа смешиваемых случайных чисел случайность, в начале, упала в 10 раз, но потом увеличилась в 100 раз. Таким образом, мы увеличили случайность данного ГПСЧ в 10 раз! Средней коэффициент накопления k=0,000048375.

Java

Перейдем к проверки накопления случайности на языке программирования Java. На практике будет использована самая новая версия языка SE 11.

Язык Java имеет 4 генератора случайных чисел:

1. Random – ГПСЧ, находящейся в составе библиотеке util, содержащей множество вспомогательных классов. Использует для генерации псевдослучайных чисел линейный конгруэнтный метод, в результате чего не является криптграфически-безопасным.
2. ThreadLocalRandom – ГПСЧ, находящейся в составе библиотеки util, отличающийся от Random тем, что изолирован от других потоков. Тоже не является криптографически-безопасным.
3. SecureRandom – ГПСЧ, введенный в язык Java для использовании в криптографии. Это аналог RNG, который использует стандартные методы

создания криптографически-стойких псевдослучайных чисел ОС, которая стоит на компьютере.

4. Match.random – ГПСЧ, используемый в библиотеке Match. По сути, вызывает Random, только результат генерации выдает в качестве числа integer, а не byte.

Генерация псевдослучайных чисел ГПСЧ Random, Match.random и ThreadLocalRandom производится линейным конгруэнтным методом – каждое новое псевдослучайное число является преобразованным предыдущим. Преобразование происходит по следующей математической функции:

$$(x \cdot 25214903917 + 11)\%2^{48}$$

Из результата берутся то 31 бит с 16 по 47. Если Вы знаете одно сгенерированное число, Вы можете предсказать следующее псевдослучайное число по данной формуле. Именно поэтому линейный конгруэнтный метод нельзя использовать в криптографии. Данные ГПСЧ реализованы так, что создают не псевдослучайные числа, а псевдослучайные байты, которые программист самостоятельно должен преобразовать в число integer.
SecureRandom является нечто вроде API для использования стандартного ГПСЧ той ОС, на которой работает программа. Является криптографически безопасным ГПСЧ.

Начнем наши испытания мы с ГПСЧ Random. Для того, чтобы использовать Random необходимо в начале программы подключить пакет java.util.Random.

```
import java.util.Random;
```

После, уже внутри функции Main необходимо создать новый ГПСЧ:

```
Random random = new Random();
```

Генерируются псевдослучайные числа класса integer, при помощи данного ГПСЧ, командой random.nextInt() с указанием в скобках верхнего предела диапазона.

```
n = random.nextInt(10);
```

Полный код программы представлен ниже:

```
//Подключаем класс Random
import java.util.Random;
```

```
//Сама программа
class Main {
    public static void main(String args[]) {

        //Создаем ГПСЧ
        Random random = new Random();

        //Создаем переменные chis
        float chis0=0;
        float chis1=0;
        float chis2=0;
        float chis3=0;
        float chis4=0;
        float chis5=0;
        float chis6=0;
        float chis7=0;
        float chis8=0;
        float chis9=0;

        //создаем переменную n для значений
        int n=0;

        //создаем переменную для подсчета количества
сгенерированных чисел
        int j=0;
```

```
        //повторяем цикл генераци и подсчета
сгенерированнх чисел
        while (j<1000000) {

            //Генерируем случайное число
            n = random.nextInt(10);

            //накапливаем случайность через XOR
            for (int i=0; i<”Количество XOR”; i++) {
                n=n^random.nextInt(10);
            }

            //Сортируем числа
            switch (n) {
                //Если сгенерированное число равно 0
                case 0:
                    chis0=chis0+1; //Увеличиваем chis0
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 1
                case 1:
                    chis1=chis1+1; //Увеличиваем chis1
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
```

```
                //Если сгенерированное число равно 2
                case 2:
                    chis2=chis2+1; //Увеличиваем chis2
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 3
                case 3:
                    chis3=chis3+1; //Увеличиваем chis3
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 4
                case 4:
                    chis4=chis4+1; //Увеличиваем chis4
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 5
                case 5:
                    chis5=chis5+1; //Увеличиваем chis5
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 6
                case 6:
                    chis6=chis6+1; //Увеличиваем chis6
на 1
```

```
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 7
                case 7:
                    chis7=chis7+1; //Увеличиваем chis7
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 8
                case 8:
                    chis8=chis8+1; //Увеличиваем chis8
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 9
                case 9:
                    chis9=chis9+1; //Увеличиваем chis9
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
            }

        }

        //выводим вероятности появления чисел от 0 до 9
        System.out.println(chis0/1000000);
        System.out.println(chis1/1000000);
```

```
            System.out.println(chis2/1000000);
            System.out.println(chis3/1000000);
            System.out.println(chis4/1000000);
            System.out.println(chis5/1000000);
            System.out.println(chis6/1000000);
            System.out.println(chis7/1000000);
            System.out.println(chis8/1000000);
            System.out.println(chis9/1000000);
        }
}
```

Результат рабоыт программы приведен в таблице ниже:

Random					
XOR	P_{max}	P_{min}	Q	S	ΔS
0	0,100437	0,099614	0,000823	0,999177	
1	0,131945	0,052608	0,079337	0,920663	–0,078514
2	0,107126	0,072535	0,034591	0,965409	+0,044746
4	0,103188	0,088211	0,014977	0,985023	+0,019614
8	0,100861	0,098109	0,002752	0,997248	+0,012225
16	0,100388	0,099707	0,000681	0,999319	+0,002071
32	0,100626	0,099438	0,001188	0,998812	–0,000507
64	0,100365	0,09965	0,000715	0,999285	+0,000473
128	0,10042	0,099303	0,001117	0,998883	–0,000402

Вывод: В результате увеличения числа смешиваемых случайных чисел случайность упала в 100 раз. При дальнейшем увеличении количества смешиваемых чисел нам удалось увеличить случайность в 100 раз, набрав максимальное значение. Стоит отетить, что случайность данного ГПСЧ изначально была максимально возможной. Средней коэффициент накопления k=−0,00003675.

Следующим ГПСЧ у нас будет ThreadLocalRandom. Для того чтобы его использовать нам необходимо подключить класс java.util.concurrent.ThreadLocalRandom:

```
import java.util.concurrent.ThreadLocalRandom;
```

Создавать ГПСЧ в качестве нового класса, как при использовании ГПСЧ Random не нужно.

Генерация случайных чисел происходит при помощи команды ThreadLocalRandom.current().nextInt() с указанием верхнего предела диапазона:

```
ThreadLocalRandom.current().nextInt(10);
```

Вот код самой программы:

```
//Подключаем класс
import java.util.concurrent.ThreadLocalRandom;


//Сама программа
class Main {
    public static void main(String args[]) {

        //Создаем переменные chis
        float chis0=0;
        float chis1=0;
        float chis2=0;
        float chis3=0;
        float chis4=0;
        float chis5=0;
        float chis6=0;
        float chis7=0;
        float chis8=0;
        float chis9=0;

        //создаем переменную n для значений
```

```
        int n=0;

        //создаем переменную для подсчета количества
сгенерированных чисел
        int j=0;

        //повторяем цикл генераци и подсчета
сгенерированнх чисел
        while (j<1000000) {

            //Генерируем случайное число
            n =
ThreadLocalRandom.current().nextInt(10);

            //накапливаем случайность через XOR
            for (int i=0; i<1; i++) {

    n=n^ThreadLocalRandom.current().nextInt(10);
            }

            //Сортируем числа
            switch (n) {
                //Если сгенерированное число равно 0
                case 0:
                    chis0=chis0+1; //Увеличиваем chis0
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
```

```
                //Если сгенерированное число равно 1
                case 1:
                    chis1=chis1+1; //Увеличиваем chis1
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 2
                case 2:
                    chis2=chis2+1; //Увеличиваем chis2
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 3
                case 3:
                    chis3=chis3+1; //Увеличиваем chis3
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 4
                case 4:
                    chis4=chis4+1; //Увеличиваем chis4
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 5
                case 5:
                    chis5=chis5+1; //Увеличиваем chis5
на 1
```

```
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 6
                case 6:
                    chis6=chis6+1; //Увеличиваем chis6
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 7
                case 7:
                    chis7=chis7+1; //Увеличиваем chis7
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 8
                case 8:
                    chis8=chis8+1; //Увеличиваем chis8
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 9
                case 9:
                    chis9=chis9+1; //Увеличиваем chis9
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
            }
```

```
            }

            //выводим вероятности появления чисел от 0 до 9
            System.out.println(chis0/1000000);
            System.out.println(chis1/1000000);
            System.out.println(chis2/1000000);
            System.out.println(chis3/1000000);
            System.out.println(chis4/1000000);
            System.out.println(chis5/1000000);
            System.out.println(chis6/1000000);
            System.out.println(chis7/1000000);
            System.out.println(chis8/1000000);
            System.out.println(chis9/1000000);
        }
}
```

Результаты работы программы сведены в таблицу ниже.

ThreadLocalRandom					
XOR	P_{max}	P_{min}	Q	S	ΔS
0	0,10042	0,099518	0,000902	0,999098	
1	0,131905	0,052472	0,079433	0,920567	–0,078531
2	0,107196	0,073051	0,034145	0,965855	+0,045288
4	0,103548	0,088454	0,015094	0,984906	+0,019051
8	0,10094	0,097682	0,003258	0,996742	+0,011836
16	0,100497	0,099433	0,001064	0,998936	+0,002194
32	0,100331	0,099527	0,000804	0,999196	+0,000260
64	0,100861	0,099551	0,00131	0,99869	–0,000506
128	0,100257	0,099324	0,000933	0,999067	+0,000377

Вывод: В результате увеличения числа смешиваемых случайных чисел случайность упала в 100 раз. При дальнейшем увеличении количества смешиваемых чисел нам удалось увеличить случайность в 100 раз, набрав

максимальное значение. Стоит отметить, что случайность данного ГПСЧ изначально была максимально возможной. Средней коэффициент накопления k=−0,000003875.
Теперь проведем эксперемент на ГПСЧ SecureRandom. Для использования данного ГПСЧ нам нужно в начале подключить класс java.security.SecureRandom:

```
java.security.SecureRandom;
```

В функции main необходимо создать ГПСЧ SecureRandom:

```
SecureRandom random = new SecureRandom();
```

Учитывая, что данный ГПСЧ создает только байты, нам придется воспользоватся массивом из 4х байтов, как в случае с ГПСЧ NGCryptoServiceProvider языка C#. К сожалению, у языка Java нет команды перевода массива в число формата integer, поэтому нам придется написать самому. Для начала, заполним массив 4 случайными битами:

```
random.nextBytes(bytes);
```

Затем нам необходимо перевести массив в число. Для этого необходимо воспользоваться циклом for. Суть перевода такова: переменную сложить со значениями всех ячеек массива по очереди. Это приведет к тому, что у нас появится в переменной число integer:

```
for (int m=0; m<4; m++) {
    n=n+bytes[m];
}
```

Как и в случае с C# мы получили число integer, диапазон которого от –2147483648 до 2147483647. Учитывая, что мы используем только положительные значения, надо сгенерированное число n взять в модуле:

```
n=Math.abs(n);
```

В результате модуля мы получили диапазон чисел от 0 до 2147483648. Данный диапазо необходимо привести к диапазону от 0 до 9 по формуле приведения диапазона:

$$x\ mod\ (n - m + 1) + m$$

У нас максимальное число 9, а минимальное 0. Таким образом у нас получается:

$$x\ mod\ (9 - 0 + 1) + 0 = x\ mod\ 10$$

В коде это выглядит так:

```
n=Math.abs(n)%10;
```

Вот полностью код генерации случайного числа:

```
random.nextBytes(bytes);
for (int m=0; m<4; m++) {
    n=n+bytes[m];
}
n=Math.abs(n)%10;
```

При накоплении случайности на XOR нам придется генерировать случайное число в переменную q, а затем уже применять функцию XOR к n и q:

```
for (int i=0; i<1; i++) {
    q=0;
    random.nextBytes(bytes);
    for (int m=0; m<4; m++) {
    q=q+bytes[m];
    }
    n=n^(Math.abs(q)%10);
}
```

Вот полный код программы:

```
//Подключаем класс SecureRandom
import java.security.SecureRandom;

//Сама программа
class Main {
```

```
    public static void main(String args[]) {

        //Создаем ГПСЧ SecureRandom
        SecureRandom random = new SecureRandom();

        //Создаем массив байтов из 4 ячеек
        byte bytes[] = new byte[4];

        //Создаем переменные chis
        float chis0=0;
        float chis1=0;
        float chis2=0;
        float chis3=0;
        float chis4=0;
        float chis5=0;
        float chis6=0;
        float chis7=0;
        float chis8=0;
        float chis9=0;

        //создаем переменную n для значений
        int n=0;

        //создаем переменную для подсчета количества
сгенерированных чисел
        int j=0;
```

```
        //Создаем дополнительную переменную для
алгоритма накопления случайности на XOR
        int q;

        //повторяем цикл генераци и подсчета
сгенерированнх чисел
        while (j<1000000) {

            //Генерируем случайное число
      random.nextBytes(bytes);
      for (int m=0; m<4; m++) {
        n=n+bytes[m];
      }
      n=Math.abs(n)%10;


            //накапливаем случайность через XOR
            for (int i=0; i<1; i++) {

        q=0;
        random.nextBytes(bytes);
        for (int m=0; m<4; m++) {
          q=q+bytes[m];
        }
        n=n^(Math.abs(q)%10);
            }
```

```
            //Сортируем числа
            switch (n) {
                //Если сгенерированное число равно 0
                case 0:
                    chis0=chis0+1; //Увеличиваем chis0
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 1
                case 1:
                    chis1=chis1+1; //Увеличиваем chis1
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 2
                case 2:
                    chis2=chis2+1; //Увеличиваем chis2
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 3
                case 3:
                    chis3=chis3+1; //Увеличиваем chis3
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
```

```
                //Если сгенерированное число равно 4
                case 4:
                    chis4=chis4+1; //Увеличиваем chis4
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 5
                case 5:
                    chis5=chis5+1; //Увеличиваем chis5
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 6
                case 6:
                    chis6=chis6+1; //Увеличиваем chis6
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 7
                case 7:
                    chis7=chis7+1; //Увеличиваем chis7
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 8
                case 8:
                    chis8=chis8+1; //Увеличиваем chis8
на 1
```

```
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 9
                case 9:
                    chis9=chis9+1; //Увеличиваем chis9
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
            }

        }

        //выводим вероятности появления чисел от 0 до 9
        System.out.println(chis0/1000000);
        System.out.println(chis1/1000000);
        System.out.println(chis2/1000000);
        System.out.println(chis3/1000000);
        System.out.println(chis4/1000000);
        System.out.println(chis5/1000000);
        System.out.println(chis6/1000000);
        System.out.println(chis7/1000000);
        System.out.println(chis8/1000000);
        System.out.println(chis9/1000000);
    }
}
```

Результаты работы программы представлены в таблице ниже:

SecureRandom					
XOR	P_{max}	P_{min}	Q	S	ΔS
0	0,10249	0,098045	0,004445	0,995555	
1	0,130885	0,051927	0,078958	0,921042	–0,074513
2	0,107738	0,071396	0,036342	0,963658	+0,042616
4	0,103336	0,087247	0,016089	0,983911	+0,020253
8	0,10091	0,097907	0,003003	0,996997	+0,013086
16	0,100434	0,099405	0,001029	0,998971	+0,001974
32	0,10037	0,099641	0,000729	0,999271	+0,000300
64	0,100607	0,09956	0,001047	0,998953	–0,000318
128	0,100304	0,099402	0,000902	0,999098	+0,000145

Вывод: В результате увеличения числа смешиваемых случайных чисел случайность упала в 10 раз. При дальнейшем увеличении количества смешиваемых чисел нам удалось увеличить случайность в 100 раз, таким образом мы увеличили случайность в 10 раз! Средней коэффициент накопления k=0,003543.

Перейдем к последнему ГПСЧ из существующих в Java – Math.random. Для его использования не требуется подключение каких-либо классов. Также не требуется создавать ГПСЧ. Вызывается данный ГПСЧ командой Math.random():

```
n=Math.random();
```

Данный ГПСЧ создает дробные случайные числа в диапазоне от 0 до 1. Это означает, что все числа имеют ноль в целой части и случайные цифры в дробной. Чтобы это исправить, необходимо сгенерированное случайное число умножить на 10. Это приведет к тому, что в целой части появится одна случайная цифра:

```
n=Math.random()*10;
```

Учитывая, что мы генерируем случайные целые числа от 0 до 9, нам необходимо отбросить дробную часть числа. Для этого необходимо преобразовать число из float в int. Это делается командой (int):

```
n=(int) (Math.random()*10);
```

Вот полностью приведенный код программы:

```
//Сама программа

class Main {

    public static void main(String args[]) {
```

```
        //Создаем переменные chis
        float chis0=0;
        float chis1=0;
        float chis2=0;
        float chis3=0;
        float chis4=0;
        float chis5=0;
        float chis6=0;
        float chis7=0;
        float chis8=0;
        float chis9=0;

        //создаем переменную n для значений
        int n=0;

        //создаем переменную для подсчета количества
сгенерированных чисел
        int j=0;

        //повторяем цикл генераци и подсчета
сгенерированнх чисел
        while (j<1000000) {

            //Генерируем случайное число
           n = (int) (Math.random()*10);
```

```
            //накапливаем случайность через XOR
            for (int i=0; i<”Количество XOR”; i++) {
                n=n^((int) (Math.random()*10));
            }

            //Сортируем числа
            switch (n) {
                //Если сгенерированное число равно 0
                case 0:
                    chis0=chis0+1; //Увеличиваем chis0
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 1
                case 1:
                    chis1=chis1+1; //Увеличиваем chis1
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 2
                case 2:
                    chis2=chis2+1; //Увеличиваем chis2
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 3
```

```
                case 3:
                    chis3=chis3+1; //Увеличиваем chis3
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 4
                case 4:
                    chis4=chis4+1; //Увеличиваем chis4
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 5
                case 5:
                    chis5=chis5+1; //Увеличиваем chis5
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 6
                case 6:
                    chis6=chis6+1; //Увеличиваем chis6
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 7
                case 7:
                    chis7=chis7+1; //Увеличиваем chis7
на 1
                    j=j+1; //Увеличиваем j на 1
```

```
                    break; //Выходим из case
                //Если сгенерированное число равно 8
                case 8:
                    chis8=chis8+1; //Увеличиваем chis8
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
                //Если сгенерированное число равно 9
                case 9:
                    chis9=chis9+1; //Увеличиваем chis9
на 1
                    j=j+1; //Увеличиваем j на 1
                    break; //Выходим из case
            }

        }

        //выводим вероятности появления чисел от 0 до 9
        System.out.println(chis0/1000000);
        System.out.println(chis1/1000000);
        System.out.println(chis2/1000000);
        System.out.println(chis3/1000000);
        System.out.println(chis4/1000000);
        System.out.println(chis5/1000000);
        System.out.println(chis6/1000000);
        System.out.println(chis7/1000000);
```

```
            System.out.println(chis8/1000000);
            System.out.println(chis9/1000000);
        }
}
```

Java					
XOR	P_{max}	P_{min}	Q	S	ΔS
0	0,107313	0,073132	0,034181	0,965819	
1	0,131888	0,052629	0,079259	0,920741	–0,045078
2	0,107847	0,072459	0,035388	0,964612	+0,043871
4	0,103329	0,088004	0,015325	0,984675	+0,020063
8	0,101224	0,098284	0,00294	0,99706	+0,012385
16	0,100686	0,09978	0,000906	0,999094	+0,002034
32	0,100585	0,099619	0,000966	0,999034	–0,000060
64	0,100514	0,099323	0,001191	0,998809	–0,000225
128	0,100543	0,099477	0,001066	0,998934	+0,000125

Вывод: В результате увеличения числа смешиваемых случайных чисел нам удалось увеличить случайность аж в 100 раз от первоначальной! Средней коэффициент накопления k=0,004139375.

JavaScript

Перейдем к испытаниям на языке JavaScript. Мы воспользуемся самым новым стандартом языка - ECMAScript 2021.

В JavaScript стандарта ECMAScript 2021 существует два ГПСЧ:

1. Match.random – стандартный ГПСЧ языка JavaScript. Создает дробное псевдослучайное число в диапазоне от нуля до еденицы, не всключая саму единицу. ($0 \leq X < 1$) . Не является криптографически безопасным.
2. Crypto.getRandomValues() – API, позовляющий использовать генератор случайных чисел самой ОС, который считается надежным для криптгографии.

Начнем мы, пожалуй, с первого ГПСЧ. Создается псевдослучайное число командой Math.random().

```
n=Math.random;
```

Данный ГПСЧ создает дробные числа, в результате чего постоянно меняется дробная часть, в части целых всегда 0. Чтобы исправить это, необходимо умножить число на 10. Тогда у нас в целых будет одна случайная цифра от 0 до 9 с дробью.

```
n=Math.random*10;
```

Для наглядности вот Вам несколько примеров:

0,123·10=1,23	0,785·10=7,85	0,587·10=5,87
0, 963·10=9,63	0,788·10=7,88	0,936·10=9,36
0,251·10=2,51	0,871·10=8,78	0,006·10=0,06
0,016·10=0,16	0,327·10=3,27	0,271·10=2,71

Нам дробь не нужна, так как мы используем только целые числа. Чтобы убрать дробь, оставив тольцо целое число, мы используем функцию Math.trunc.

```
n=Math.trunc(Math.random()*10);
```

Полностью код программы показан ниже:

```
//начало программы

<Script>


//Создаем переменные chis

var chis0=0;

var chis1=0;

var chis2=0;

var chis3=0;

var chis4=0;

var chis5=0;

var chis6=0;

var chis7=0;

var chis8=0;

var chis9=0;


//создаем переменную n для значений

var n=0;
```

```
//создаем переменную для подсчета количества
сгенерированных чисел
var j=0;

//повторяем цикл генераци и подсчета сгенерированнх
чисел
    while (j<1000000) {

//Генерируем случайное число
        n=Math.trunc(Math.random()*10);

//накапливаем случайность через XOR
        for (var i=0; i<”Количество XOR”; i++) {
            n=n^Math.trunc(Math.random()*10);
        }

//Сортируем числа
switch (n) {
    //Если сгенерированное число равно 0
    case 0:
        chis0=chis0+1; //Увеличиваем chis0 на 1
        j=j+1; //Увеличиваем j на 1
        break; //Выходим из case
    //Если сгенерированное число равно 1
        case 1:
```

```
            chis1=chis1+1; //Увеличиваем chis1 на 1
            j=j+1; //Увеличиваем j на 1
            break; //Выходим из case
    //Если сгенерированное число равно 2
        case 2:
            chis2=chis2+1; //Увеличиваем chis2 на 1
            j=j+1; //Увеличиваем j на 1
            break; //Выходим из case
    //Если сгенерированное число равно 3
        case 3:
            chis3=chis3+1; //Увеличиваем chis3 на 1
            j=j+1; //Увеличиваем j на 1
            break; //Выходим из case
    //Если сгенерированное число равно 4
        case 4:
            chis4=chis4+1; //Увеличиваем chis4 на 1
           j=j+1; //Увеличиваем j на 1
           break; //Выходим из case
    //Если сгенерированное число равно 5
        case 5:
            chis5=chis5+1; //Увеличиваем chis5 на 1
            j=j+1; //Увеличиваем j на 1
            break; //Выходим из case
    //Если сгенерированное число равно 6
        case 6:
            chis6=chis6+1; //Увеличиваем chis6 на 1
```

```
        j=j+1; //Увеличиваем j на 1
        break; //Выходим из case
//Если сгенерированное число равно 7
    case 7:
        chis7=chis7+1; //Увеличиваем chis7 на 1
        j=j+1; //Увеличиваем j на 1
        break; //Выходим из case
//Если сгенерированное число равно 8
    case 8:
        chis8=chis8+1; //Увеличиваем chis8 на 1
        j=j+1; //Увеличиваем j на 1
        break; //Выходим из case
//Если сгенерированное число равно 9
    case 9:
        chis9=chis9+1; //Увеличиваем chis9 на 1
        j=j+1; //Увеличиваем j на 1
        break; //Выходим из case
    }
}

//выводим вероятности появления чисел от 0 до 9
document.write("<p>"+chis0/1000000+"</p>");
document.write("<p>"+chis1/1000000+"</p>");
document.write("<p>"+chis2/1000000+"</p>");
document.write("<p>"+chis3/1000000+"</p>");
document.write("<p>"+chis4/1000000+"</p>");
```

```
    document.write("<p>"+chis5/1000000+"</p>");
    document.write("<p>"+chis6/1000000+"</p>");
    document.write("<p>"+chis7/1000000+"</p>");
    document.write("<p>"+chis8/1000000+"</p>");
    document.write("<p>"+chis9/1000000+"</p>");
</script>
```

ECMAScript					
XOR	P_{max}	P_{min}	Q	S	ΔS
0	0,100562	0,099153	0,001409	0,998591	
1	0,100211	0,039947	0,060264	0,939736	–0,058855
2	0,076721	0,051964	0,024757	0,975243	+0,035507
4	0,06788	0,057429	0,010451	0,989549	+0,014306
8	0,063443	0,061636	0,001807	0,998193	+0,008644
16	0,062707	0,062099	0,000608	0,999392	+0,001199
32	0,063049	0,061962	0,001087	0,998913	–0,000479
64	0,062876	0,062149	0,000727	0,999273	+0,000360
128	0,062692	0,062225	0,000467	0,999533	+0,000260

Вывод: В результате увеличения числа смешиваемых случайных чисел случайность упала в 10 раз. При дальнейшем увеличении количества смешиваемых чисел нам удалось увеличить случайность в 100 раз, таким образом мы увеличили случайность в 10 раз! Средней коэффициент накопления k=0,00011775.

Перейдем к испытанию второго ГПСЧ, который доступен в JavaScript через API - Web Cryptography API.

Данный ГПСЧ не создает случайных чисел integer, а только наполянет случайными байтами массивы как ГПСЧ SecureRandom языка Java и NGCryptoServiceProvider языка C#. Собственно говоря, это происходит потому, что все это API к одному и тому же ГПСЧ, который установлен в моей ОС. Однако, в отличие от SecureRandom и NGCryptoServiceProvider, в языке JavaScriptмы можем использовать массив с одной ячейкой, размером 32 бита (4 байта), которую преобразуем в число integer.

Создается такой массив так:

```
var array = new Uint32Array(1);
```

Теперь нам нужно наполнить данный массив случайными числами:

```
window.crypto.getRandomValues(array);
```

Чтобы перевести данные массива в число integer, мы воспользуемся командой parseInt(), которая преобразет ячейку массива в число с указанным основанием системы счисления. В нашем случае это 10:

```
n=parseInt(window.crypto.getRandomValues(array), 10);
```

В результате данных преобразований, мы получили случайное число в диапазоне от -2147483648 до 2147483647. Но нам нужно от 0 до 9, поэтому приведем его к нужному диапазону, взяв остаток от деления на 10:

```
n=(parseInt(window.crypto.getRandomValues(array),
10))%10;
```

Ниже представлен код всей программы:

```
//начало программы

<Script>


//Создаем переменные chis

var chis0=0;

var chis1=0;

var chis2=0;

var chis3=0;

var chis4=0;

var chis5=0;

var chis6=0;

var chis7=0;

var chis8=0;

var chis9=0;


//создаем переменную n для значений
```

```
var n=0;


//создаем переменную для подсчета количества
сгенерированных чисел
var j=0;


//создаем массив для генерации случайных чисел
var array = new Uint32Array(1);


//повторяем цикл генераци и подсчета сгенерированнх
чисел
    while (j<1000000) {



//Генерируем случайное число

    n=(parseInt(window.crypto.getRandomValues(array),
10))%10;


//накапливаем случайность через XOR
        for (var i=0; i<0; i++) {

    n=n^((parseInt(window.crypto.getRandomValues(array)
, 10))%10);
        }


//Сортируем числа
switch (n) {
```

```
//Если сгенерированное число равно 0
case 0:
    chis0=chis0+1; //Увеличиваем chis0 на 1
    j=j+1; //Увеличиваем j на 1
    break; //Выходим из case
//Если сгенерированное число равно 1
    case 1:
        chis1=chis1+1; //Увеличиваем chis1 на 1
        j=j+1; //Увеличиваем j на 1
        break; //Выходим из case
//Если сгенерированное число равно 2
    case 2:
        chis2=chis2+1; //Увеличиваем chis2 на 1
        j=j+1; //Увеличиваем j на 1
        break; //Выходим из case
//Если сгенерированное число равно 3
    case 3:
        chis3=chis3+1; //Увеличиваем chis3 на 1
        j=j+1; //Увеличиваем j на 1
        break; //Выходим из case
//Если сгенерированное число равно 4
    case 4:
        chis4=chis4+1; //Увеличиваем chis4 на 1
        j=j+1; //Увеличиваем j на 1
        break; //Выходим из case
//Если сгенерированное число равно 5
```

```
    case 5:
        chis5=chis5+1; //Увеличиваем chis5 на 1
        j=j+1; //Увеличиваем j на 1
        break; //Выходим из case
//Если сгенерированное число равно 6
    case 6:
        chis6=chis6+1; //Увеличиваем chis6 на 1
        j=j+1; //Увеличиваем j на 1
        break; //Выходим из case
//Если сгенерированное число равно 7
    case 7:
        chis7=chis7+1; //Увеличиваем chis7 на 1
        j=j+1; //Увеличиваем j на 1
        break; //Выходим из case
//Если сгенерированное число равно 8
    case 8:
        chis8=chis8+1; //Увеличиваем chis8 на 1
        j=j+1; //Увеличиваем j на 1
        break; //Выходим из case
//Если сгенерированное число равно 9
    case 9:
        chis9=chis9+1; //Увеличиваем chis9 на 1
        j=j+1; //Увеличиваем j на 1
        break; //Выходим из case
    }
}
```

```
    //выводим вероятности появления чисел от 0 до 9
    document.write("<p>"+chis0/1000000+"</p>");
    document.write("<p>"+chis1/1000000+"</p>");
    document.write("<p>"+chis2/1000000+"</p>");
    document.write("<p>"+chis3/1000000+"</p>");
    document.write("<p>"+chis4/1000000+"</p>");
    document.write("<p>"+chis5/1000000+"</p>");
    document.write("<p>"+chis6/1000000+"</p>");
    document.write("<p>"+chis7/1000000+"</p>");
    document.write("<p>"+chis8/1000000+"</p>");
    document.write("<p>"+chis9/1000000+"</p>");
</script>
```

Web Cryptography API.					
XOR	P_{max}	P_{min}	Q	S	ΔS
0	0,100755	0,099366	0,001389	0,998611	
1	0,131638	0,052604	0,079034	0,920966	–0,077645
2	0,107081	0,073092	0,033989	0,966011	+0,045045
4	0,103402	0,087917	0,015485	0,984515	+0,018504
8	0,100984	0,098461	0,002523	0,997477	+0,012962
16	0,10042	0,099682	0,000738	0,999262	+0,001785
32	0,100665	0,099461	0,001204	0,998796	–0,000466
64	0,100415	0,099525	0,00089	0,99911	+0,000314
128	0,10047	0,099331	0,001139	0,998861	–0,000249

Вывод: В результате увеличения числа смешиваемых случайных чисел случайность упала в 10 раз. При дальнейшем увеличении количества смешиваемых чисел нам удалось увеличить случайность в 100 раз, таким образом мы увеличили случайность в 10 раз! Средней коэффициент накопления k=0,00003125.

Pascal

Для проведения эксперемента используется стандарт ISO. В качестве компилятора была выбрана последняя версия компилятора PascalABC.NET версии 3.8.2.

В языке программирования Pascal еть только один ГПСЧ, который вызывается командой random() с указанием в скобках через запятую диапазона чисел. В нашем случае это от 0 до 9:

```
n:=random(0, 9);
```

Стоит отметить, что в языке Pascal вместо знака «=» требуется ставить «:=», а вместо «^» (команда XOR) команду «XOR»:

```
n:=n XOR random(0, 9);
```

Также вместо фигурных скобок «{ } » в Pascal используются ключевые слова «begin» и «end».

Полностью программа представлена ниже:

```
//Создаем переменные chis
var chis0: integer;
var chis1: integer;
var chis2: integer;
var chis3: integer;
var chis4: integer;
var chis5: integer;
var chis6: integer;
var chis7: integer;
var chis8: integer;
var chis9: integer;

//создаем переменную n для значений
var n:integer;

//создаем переменную для подсчета количества
сгенерированных чисел
var j: integer;

//начало программы
begin

    //обнуляем счетчик j
    j:=0;

    //повторяем цикл генераци и подсчета сгенерированнх
чисел
    while j<1000000 do begin

    //Генерируем случайное число
    n:=random(0, 9);
```

```
//накапливаем случайность через XOR
for var i:= 0 to "Количество XOR" do begin
    n:=n XOR random(0, 9);
end;

//Сортируем числа
case n of
    //Если сгенерированное число равно 0
    0: begin
        chis0:=chis0+1; //Увеличиваем chis0 на 1
        j:=j+1; //Увеличиваем j на 1
        end;
    //Если сгенерированное число равно 1
    1: begin
        chis1:=chis1+1; //Увеличиваем chis0 на 1
        j:=j+1; //Увеличиваем j на 1
    end;
    //Если сгенерированное число равно 2
    2: begin
        chis2:=chis2+1; //Увеличиваем chis0 на 1
        j:=j+1; //Увеличиваем j на 1
    end;
    //Если сгенерированное число равно 3
    3: begin
        chis3:=chis3+1; //Увеличиваем chis0 на 1
        j:=j+1; //Увеличиваем j на 1
    end;
    //Если сгенерированное число равно 4
    4: begin
        chis4:=chis4+1; //Увеличиваем chis0 на 1
        j:=j+1; //Увеличиваем j на 1
    end;
    //Если сгенерированное число равно 5
    5: begin
        chis5:=chis5+1; //Увеличиваем chis0 на 1
        j:=j+1; //Увеличиваем j на 1
    end;
    //Если сгенерированное число равно 6
    6: begin
        chis6:=chis6+1; //Увеличиваем chis0 на 1
        j:=j+1; //Увеличиваем j на 1
    end;
    //Если сгенерированное число равно 7
    7: begin
```

```
            chis7:=chis7+1; //Увеличиваем chis0 на 1
            j:=j+1; //Увеличиваем j на 1
        end;
        //Если сгенерированное число равно 8
        8: begin
            chis8:=chis8+1; //Увеличиваем chis0 на 1
            j:=j+1; //Увеличиваем j на 1
        end;
        //Если сгенерированное число равно 9
        9: begin
            chis9:=chis9+1; //Увеличиваем chis0 на 1
            j:=j+1; //Увеличиваем j на 1
        end;
    end;
    end;

    //выводим вероятности появления чисел от 0 до 9
    println('0: ', chis0, chis0/1000000);
    println('1: ', chis1, chis1/1000000);
    println('2: ', chis2, chis2/1000000);
    println('3: ', chis3, chis3/1000000);
    println('4: ', chis4, chis4/1000000);
    println('5: ', chis5, chis5/1000000);
    println('6: ', chis6, chis6/1000000);
    println('7: ', chis7, chis7/1000000);
    println('8: ', chis8, chis8/1000000);
    println('9: ', chis9, chis9/1000000);
end.
```

Pascal					
XOR	P_{max}	P_{min}	Q	S	ΔS
0	0,100397	0,099299	0,001098	0,998902	
1	0,099665	0,039906	0,059759	0,940241	–0,058661
2	0,076541	0,051582	0,024959	0,975041	+0,034800
4	0,067944	0,057534	0,01041	0,98959	+0,014549
8	0,063561	0,062171	0,00139	0,99861	+0,009020
16	0,06302	0,062072	0,000948	0,999052	+0,000442
32	0,062762	0,062344	0,000418	0,999582	+0,000530
64	0,062917	0,06197	0,000947	0,999053	–0,000529
128	0,062768	0,062119	0,000649	0,999351	+0,000298

Вывод: В результате увеличения числа смешиваемых случайных чисел случайность упала в 10 раз. При дальнейшем увеличении количества смешиваемых чисел нам удалось увеличить случайность в 100 раз, таким

образом мы увеличили случайность в 10 раз! Средней коэффициент накопления k=0,000056125.

BASIC

На практике мы воспользуемся компилятором языка BASIC под названием Liberty последней версии (Liberty BASIC 4.5.1).

В языке программирования BASIC, как и в Pascal, вместо «^» используется комманда «XOR». Также функция взятия остатка от деления обозначается командой MOD, а не знаком процента (%). В конце строки не ставится точка с запятой (;). Сами же команды принято писать заглавными буквами.

ГПСЧ языка BASIC очень похож на ГПСЧ языков C и C++. Он, также как и они, берет число, а затем путем математических преобразований «превращает его в случайное число». Запускается ГПСЧ командой RND с указанием в скобке любого числа, чаще всего 1. Это число вообще никак не влияет на работу ГПСЧ.

```
N=RND(1)
```

Чтобы увеличить случайность необходимо каждый раз устанавливать новое начальное число ГПСЧ. Для этого используется функция RANDMOMIZE с указанием начального числа (мы поставим 1). Сама же функция должна быть перед командой RND:

```
RANDOMIZE 1

N=RND(1)
```

Если число RANDOMIZE не менять, ГПСЧ будет каждый раз выдавать одно и то же число. Чтобы не вводить пользователю каждый раз новое число, мы, как и в примерах с C и C++, будем использовать системное время. Для этого в языке программирования BASIC есть команда time$(“ms”), которая выводит время, установленное на компьютере, в миллисекундах.

```
time$("ms")
```

RANDIMIZE должен принимать значение меньше 1 и больше 0, при этом иметь не более двух знаков после запятой, в то время как time$ выдает большое целое число. Чтобы это исправить, нам необходимо функцию time$ привести к нужному диапазону: 0<X<1. Для этого давайте воспользуемся нашей формулой приведения:

$$x\ mod\ (n-m)+m=\text{c}$$

Здесь x – значение функции time$, n – верхняя граница диапазона, m –нижняя граница диапазона. Учитывая, что результат функции time$ должен быть в пределах 0<X<1 с двумя знаками после запятой, мы получаем диапазон от 0,01 до 0,99. Обратите внимание! Наиболее близкое число к 0,99 – это 1. Таким образом, в данном примере, m=0,01 и n=1.

Подставляем наши значения в формулу и вычисляем:

$$x\ mod\ (1 - 0{,}01) + 0{,}01 = x\ mod\ 0{,}99 + 0{,}01$$

Теперь произведем данные действия с командой time$:

```
time$("ms") mod (0.99)+0.01
```

Эту строчку кода мы должны использовать в качестве аргумента функции RANDOMIZE:

```
RANDOMIZE time$("ms") mod (0.99)+0.01

N=RND(1)
```

Теперь функция RND сохраняет в переменную N разные псевдослучайные дробные числа. Чтобы сделать из них целые числа нам понадобится как и в языке JavaScript умножить сгенерированное число на 10, а за тем отбросить дробь.

Для избавления от дробной части применим команду INT():

```
N=INT(RND(1)*10)
```

В общем виде это выглядит вот так:

```
RANDOMIZE time$("ms") MOD (0.98)+0.01

N=INT(RND(1)*10)
```

Вот код программы:

```
REM Создаем переменные chis

chis0=0

chis1=0

chis2=0
```

```
chis3=0
chis4=0
chis5=0
chis6=0
chis7=0
chis8=0
chis9=0

REM создаем переменную для подсчета количества сгенерированных чисел
J=0

REM повторяем цикл генераци и подсчета сгенерированнх чисел
WHILE (J<1000)

    REM Генерируем случайное число
    RANDOMIZE time$("ms") MOD (0.99)+0.01
    N=INT(RND(1)*10)

    REM накапливаем случайность через XOR
    FOR I=1 TO “Количество XOR” STEP 1
        RANDOMIZE time$("ms") MOD (0.99)+0.01
        N=N XOR INT(RND(1)*10)
    NEXT
```

```
    REM Сортируем числа
    SELECT CASE N
    REM Если сгенерировано число 0, то увеличиваем chis0 и j на единицу.
        CASE 0
            chis0=chis0+1
            J=J+1
        REM Если сгенерировано число 1, то увеличиваем chis1 и j на единицу.
        CASE 1
            chis1=chis1+1
            J=J+1
        REM Если сгенерировано число 2, то увеличиваем chis2 и j на единицу.
        CASE 2
            chis2=chis2+1
            J=J+1
        REM Если сгенерировано число 3, то увеличиваем chis3 и j на единицу.
        CASE 3
            chis3=chis3+1
            J=J+1
        REM Если сгенерировано число 4, то увеличиваем chis4 и j на единицу.
        CASE 4
            chis4=chis4+1
            J=J+1
```

```
        REM Если сгенерировано число 5, то увеличиваем
    chis5 и j на единицу.
        CASE 5
            chis5=chis5+1
            J=J+1
        REM Если сгенерировано число 6, то увеличиваем
    chis6 и j на единицу.
        CASE 6
            chis6=chis6+1
            J=J+1
        REM Если сгенерировано число 7, то увеличиваем
    chis7 и j на единицу.
        CASE 7
            chis7=chis7+1
            J=J+1
        REM Если сгенерировано число 8, то увеличиваем
    chis8 и j на еденицу.
        CASE 8
            chis8=chis8+1
            J=J+1
        REM Если сгенерировано число 9, то увеличиваем
    chis9 и j на еденицу.
        CASE 9
            chis9=chis9+1
            J=J+1
        END SELECT
WEND
```

```
REM выводим вероятности появления чисел от 0 до 9
PRINT chis0/1000000
PRINT chis1/1000000
PRINT chis2/1000000
PRINT chis3/1000000
PRINT chis4/1000000
PRINT chis5/1000000
PRINT chis6/1000000
PRINT chis7/1000000
PRINT chis8/1000000
PRINT chis9/1000000

END
```

Результаты работы программы:

BASIC					
XOR	P_{max}	P_{min}	Q	S	ΔS
0	0,121361	0,077387	0,043974	0,956026	
1	0,995178	0,000005	0,995173	0,004827	–0,951199
2	0,115375	0,078857	0,036518	0,963482	+0,958655
4	0,114474	0,079677	0,034797	0,965203	+0,001721
8	0,110926	0,077611	0,033315	0,966685	+0,001482
16	0,112544	0,082007	0,030537	0,969463	+0,002778
32	0,111304	0,08223	0,029074	0,970926	+0,001463
64	0,112156	0,080792	0,031364	0,968636	–0,002290
128	0,110898	0,080195	0,030703	0,969297	+0,000661

Вывод: В результате увеличения числа смешиваемых случайных чисел случайность упала в 10 раз. При дальнейшем увеличении количества смешиваемых чисел нам удалось увеличить случайность, но 128 случайных чисел недостаточно, чтобы превзойти начальный уровень случайности. Средней коэффициент накопления k=0,013271

Сравнение результатов

Мы провели накопление на 17 различных ГПСЧ. Для каждого из них мы высчитали все необходимые параметры и свели в таблицу. Настало время сравнить результаты и сделать выводы. Для этого мы должны ответить на 2 вопроса:

1. Работает ли наша теория накопления случайности на практике?
2. На любом ли ГПСЧ возможно накопление случайности?

На первый вопрос мы сможем ответить, просчитав разницу между начальной случайностью (случайностью самого ГПСЧ без накопления, S_1) и конечной случайностью (случайностью при смешении 128 случайных чисел этого ГПСЧ, S_8). Расчет будет происходить по формуле:

$$a = S_8 - S_1$$

где: a – количество накопленной случайности, S_8 – случайность, указанная в 8 строке таблицы (при смешивании 128 чисел), S_1 – случайность, указанная в первой строке таблицы (без смешивания чисел, значение самого ГПСЧ).

На второй вопрос мы сможем ответить, рассчитав разницу между случайностью при смешивании 128 чисел (8 строка, S_8) и одном смешивании (строка 2, S_2). Расчет проходит по формуле:

$$a = S_8 - S_2$$

где: a – количество накопленной случайности, S_8 – случайность, указанная в 8 строке таблицы (при смешивании 128 чисел), S2 – случайность, указанная во второй строке таблицы (при смешивании с 1 псевдослучайным числом).

Кроме того, сравнив средний коэффициент накопления случайности (k), мы сможем определить самый удобный для накопления ГПСЧ.

Результаты расчетов, а также средний коэффициент накопления случайности (k) сведены в таблицу ниже:

Язык	ГПСЧ	S_8- S_1	S_8- S_2	k
PHP	Вихрь Марсенна	+0,000177	+0,059691	+0,00000142968
	CryptGenRandom	+0,000127	+0,059795	+0,000015875
	CNG	+0,003783	+0,059973	+0,000472875
	md_rand	+0,003175	+0,059716	+0,000396875
	Генератор уникальных ID	−0,000302	+0,998751	−0,00003775
C	ANSI C	+0,454075	+0,292923	+0,056759375
C++	C++20	+0,678064	+0,441618	+0,084758
C#	ISO C#	+0,110395	+0,100864	+0,013799375

	RNGCryptoServiceProvider	+0,000387	+0,078098	+0,000048375
Java	Random	−0,000294	+0,07822	−0,00003675
	ThreadLocalRandom	−0,000031	+0,0785	−0,000003875
	SecureRandom	+0,003543	+0,078056	+0,003543
	Match.random	+0,033115	+0,078193	+0,004139375
JavaScript	ECMAScript	+0,000942	+0,059797	+0,00011775
	Web Cryptography API	+0,00025	+0,077895	+0,00003125
Pascal	Pascal	+0,000449	+0,05911	+0,000056125
BASIC	BASIC	+0,013271	+0,96447	+0,013271

Выводы:

1. Нам удалось накопить случайность почти на всех ГПСЧ. Исключением стали только 4: Random, ThreadLocalRandom языка JavaScript, Генератор уникальных ID языка PHP. Но это связано с тем, что требуется смешения большего числа сгенерированных чисел.
2. Накопление случайности возможно на любом ГПСЧ

Быстрее всего накапливал случайность ГПСЧ языка C++. Медленнее всего накапливал случайность генератор ID.