3.6. ВЫВОДЫ

В ЭТОЙ ГЛАВЕ было рассмотрено довольно много тем: генерирование случайных чисел, их проверка, их видоизменение при использовании и методы получения теоретических фактов. Возможно, главным для многих читателей был вопрос "Что получено в результате всей этой теории и что такое простой добротный генератор, который можно использовать в программах в качестве надежного источника случайных чисел?".

Подробные исследования в этой главе наводят на мысль, что следующая процедура позволяет получить простейший генератор случайных чисел для машинного языка большинства компьютеров. В начале программы присвойте целой переменной X некоторое значение X_0. Эта величина X используется только для генерирования случайного числа. Как только в программе потребуется новое случайное число, положите

$$X \leftarrow (aX + c) \bmod m \tag{1}$$

и используйте новое значение X в качестве случайной величины. Необходимо тщательно выбрать X_0, a, c и m и разумно использовать случайные числа согласно следующим принципам.

i) "Начальное" число X_0 может быть выбрано произвольно. Если программа используется несколько раз и каждый раз требуются различные источники случайных чисел, то нужно присвоить X_0 последнее полученное значение X на предыдущем прогоне или, если это более удобно, присвоить X_0 текущие дату и время. Чтобы снова запустить программу с *такими же* случайными числами (например, при отладке программы), нужно напечатать X_0, если иначе его невозможно получить.

ii) Число m должно быть большим, скажем, по крайней мере 2^{30}. Возможно, удобно взять его равным размеру компьютерного слова, так как это делает вычисление $(aX + c) \bmod m$ вполне эффективным. В разделе 3.2.1.1 выбор m обсуждается более детально. Вычисление $(aX + c) \bmod m$ должно быть *точным* без округления ошибки.

iii) Если m — степень 2 (т. е. если используется двоичный компьютер), выбираем a таким, чтобы $a \bmod 8 = 5$. Если m — степень 10 (т. е. используется десятичный компьютер), выбираем a таким, чтобы $a \bmod 200 = 21$. Одновременный выбор a и c даст гарантию, что генератор случайных чисел будет вырабатывать все m различных возможных значений X прежде, чем они начнут повторяться (см. раздел 3.2.1.2), и гарантирует высокий "потенциал" (см. раздел 3.2.1.3).

iv) Множитель a предпочтительно выбирать между $.01m$ и $.99m$, и его двоичные или десятичные цифры *не* должны иметь простую регулярную структуру. Выбирая несколько случайных констант, подобных $a = 3141592621$ (которые удовлетворяют обоим условиям в (iii)), почти всегда получаем достаточно хороший множитель. Дополнительная проверка, конечно, нужна, если генератор случайных чисел используется регулярно. Например, частичные отношения не должны быть большими, когда для нахождения $\gcd$ a и m используется алгоритм Евклида (см. раздел 3.3.3). Множитель должен пройти спектральный критерий (раздел 3.3.4) и несколько критериев, описанных в разделе 3.3.2, прежде чем он получит сертификат качества.

v) Значение c не существенно, когда a — хороший множитель, за исключением того, что c не должно иметь общего множителя с m, когда m — размер компьютерного слова. Таким образом, можно выбрать $c = 1$ или $c = a$. Многие используют $c = 0$ вместе с $m = 2^e$, но они жертвуют двумя двоичными разрядами точности и половиной начальных значений, чтобы сэкономить всего несколько наносекунд счета (см. упр. 3.2.1.2–9).

vi) Младшие значащие цифры (справа) X не очень случайны, так что решения, основанные на числе X, всегда должны опираться, главным образом, на старшие значащие цифры. Обычно лучше считать X случайной дробью X/m между 0 и 1, т. е. представлять себе X с десятичной точкой слева, а не относиться к X как к случайному целому числу между 0 и $m - 1$. Чтобы подсчитать случайное целое число между 0 и $k - 1$, нужно умножить его на k (но не делить на k; см. упр. 3.4.1–3) и округлить результат.

vii) Важное ограничение случайности последовательности (1) обсуждалось в разделе 3.3.4, в котором показано, что "точность" при размерности t будет только порядка $\sqrt[t]{m}$. Применяя метод Монте-Карло, необходимо использовать случайные последовательности высокой надежности. Их можно получить с помощью технических приемов, описанных в разделе 3.2.2.

viii) Можно генерировать не больше $m/1000$ чисел, иначе последующие будут вести себя подобно предыдущим. Если $m = 2^{32}$, значит, новая схема (например, новый множитель a) должна использоваться после генерирования нескольких миллионов случайных чисел.

Комментарии, приведенные выше, относятся, главным образом, к машинному языку программирования. Некоторые понятия прекрасно работают также на языках высокого уровня, например (1) превращается в `X=a*X+c` на языке C, если `X` имеет тип `беззнаковое длинное` и если m равно модулю `беззнаково длинной` арифметики (обычно 2^{32} или 2^{64}). Но C не дает возможности рассматривать `X` в качестве дроби, как того требует (vi), если не обращаться к числам с плавающей точкой двойной точности.

Поэтому для языка программирования, подобного C, часто используют другой вариант (1): выбирается простое число m, близкое к наибольшему легко вычисляемому целому числу, a полагается равным первообразному корню m и приращение c в этом случае равняется нулю. Тогда (1) может полностью выполняться простой арифметикой над числами, остающимися между $-m$ и $+m$, так, как в упр. 3.2.1.1–9. Например, когда $a = 48271$ и $m = 2^{31} - 1$ (см. строку 20 табл. 3.3.4–1), можно вычислить $X \leftarrow aX \bmod m$ на языке C.

```
#define MM 2147483647              /* простое число Мерсена */
#define AA 48271  /* здесь хорош спектральный критерий */
#define QQ 44488  /* (long)(MM/AA) */
#define RR 3399   /* MM % AA; важно, что RR<QQ */
X=AA*(X%QQ)-RR*(long)(X/QQ);
if (X<0) X+=MM;
```

Здесь `X` имеет тип `long` и `X` можно инициализировать, как ненулевое начальное значение, меньшее `MM`. Поскольку `MM` — простое число, наименьший значащий дво-

ичный разряд X так же случаен, как и самый старший, поэтому в предостережении (vi) нет необходимости.

Чтобы получить миллионы и миллионы случайных чисел, можно скомбинировать эту программу с другой, как в 3.3.4–(38), дописав дополнительно несколько операций.

```
#define MMM 2147483399        /* простое число, но не Мерсена */
#define AAA 40692       /* другой удачный спектральный критерий */
#define QQQ 52774       /* (long)(MMM/AAA) */
#define RRR 3791        /* MMM % AAA; снова меньше, чем QQQ */
Y=AAA*(Y%QQQ)-RRR*(long)(Y/QQQ);
if (Y<0) Y+=MMM;
Z=X-Y; if (Z<=0) Z+=MM;
```

Подобно X, случайная величина Y должна быть установлена не равной нулю. Эти операции несколько отличаются от операций, приведенных в 3.3.4–(38): выходное Z никогда не равно нулю и Z всегда лежит точно между 0 и 2^{31}. Длина периода последовательности Z приблизительно равна 74 квадриллионам, и ее числа сейчас имеют точность, приблизительно вдвое бóльшую, чем числа X.

Этот метод мобильный и совершенно простой, но не очень устойчивый. Альтернативная схема, основанная на последовательности Фибоначчи с запаздыванием и вычитанием (упр. 3.2.2–23), даже более привлекательна, так как не только легко преобразуется для использования на разных компьютерах, но значительно быстрее работает и поставляет случайные числа лучшего качества, поскольку t-мерная точность, возможно, хороша для $t \le 100$. Ниже приведена программа на языке C *ran_array*(**long** *aa*[], **int** *n*), которая генерирует n новых случайных чисел и засылает их в заданный массив *aa*, используя рекуррентное соотношение

$$X_j = (X_{j-100} - X_{j-37}) \bmod 2^{30}. \tag{2}$$

Это соотношение, в частности, хорошо подходит для современных компьютеров. Значение n должно быть равно по крайней мере 100; рекомендуются большие значения, например 1 000.

```
#define KK 100                          /* длинное запаздывание */
#define LL  37                          /* короткое запаздывание */
#define MM (1L<<30)                        /* модуль */
#define mod_diff(x,y) (((x)-(y))&(MM-1))    /* (x-y) mod MM */
long ran_x[KK];                        /* состояние генератора */
void ran_array(long aa[],int n) {
  register int i,j;
  for (j=0;j<KK;j++) aa[j]=ran_x[j];
  for (;j<n;j++) aa[j]=mod_diff(aa[j-KK],aa[j-LL]);
  for (i=0;i<LL;i++,j++) ran_x[i]=mod_diff(aa[j-KK],aa[j-LL]);
  for (;i<KK;i++,j++) ran_x[i]=mod_diff(aa[j-KK],ran_x[i-LL]);
}
```

Вся информация о числах, которые генерируются путем заблаговременного обращения к *ran_array*, появляется в массиве *ran_x*. Так, этот массив можно ско-

пировать во время вычислений, чтобы позднее повторить запуск программы с такими же значениями, не проходя весь путь назад, к началу последовательности. Особенностью использования рекуррентных соотношений, подобных (2), является, конечно, необходимость получения сразу всех начальных значений путем присвоения подходящих значений $X_0, \ldots, X_{99}$. Следующая программа *ran_start*(**long** *seed*) хорошо запускает генератор, когда задано любое начальное число, находящееся между 0 и $2^{30} - 3 = 1{,}073{,}741{,}821$ включительно.

```
#define TT  70     /* гарантирует разделение потоков */
#define is_odd(x)  ((x)&1)            /* блок двоичных разрядов x */
#define evenize(x) ((x)&(MM-2))              /* делаем x четным */

void ran_start(long seed) { /* используется для размещения ran_array */
  register int t,j;
  long x[KK+KK-1];                 /* подготовка буфера */
  register long ss=evenize(seed+2);
  for (j=0;j<KK;j++) {
    x[j]=ss;                          /* самозагрузка буфера */
    ss<<=1; if (ss>=MM) ss-=MM-2;    /* циклический сдвиг 29 двоичных
                                        разрядов */
  }
  for (;j<KK+KK-1;j++) x[j]=0;
  x[1]++;                    /* делаем x[1] (и только x[1]) нечетным */
  ss=seed&(MM-1); t=TT-1; while (t) {
    for (j=KK-1;j>0;j--) x[j+j]=x[j];           /* "квадрат" */
    for (j=KK+KK-2;j>KK-LL;j-=2) x[KK+KK-1-j]=evenize(x[j]);
    for (j=KK+KK-2;j>=KK;j--) if(is_odd(x[j])) {
      x[j-(KK-LL)]=mod_diff(x[j-(KK-LL)],x[j]);
      x[j-KK]=mod_diff(x[j-KK],x[j]);
    }
    if (is_odd(ss)) {                    /* "умножаем на z" */
      for (j=KK;j>0;j--)  x[j]=x[j-1];
      x[0]=x[KK];            /* сдвигаем буфер циклично */
      if (is_odd(x[KK])) x[LL]=mod_diff(x[LL],x[KK]);
    }
    if (ss) ss>>=1; else t--;
  }
  for (j=0;j<LL;j++) ran_x[j+KK-LL]=x[j];
  for (;j<KK;j++) ran_x[j-LL]=x[j];
}
```

Довольно любопытное маневрирование *ran_start* приведено в упр. 9, в котором доказывается, что последовательности чисел, генерируемых с различными начальными значениями, независимы: *каждый блок 100 последовательных значений* X_n, X_{n+1}, $\ldots$, X_{n+99} *в последующем выходном массиве ran_array будет отличаться от блоков, появляющихся с другим начальным значением.* (Строго говоря, известно, что это справедливо только тогда, когда $n < 2^{70}$, но меньше 2^{55} нс в год.) Некоторые процессы можно, следовательно, запускать параллельно с различными начальными

значениями и быть уверенным, что они дадут независимые результаты. Разные группы ученых, работающих над задачей в различных компьютерных центрах, могут быть уверены, что они не дублируют работу других ученых, если присваивают различные начальные значения. Таким образом, более одного миллиарда практически непересекающихся групп случайных чисел предусмотрено единственными программами *ran_array* и *ran_start*. А если этого недостаточно, можете заменить параметры программы 100 и 37 другими значениями из табл. 3.2.2–1.

В программах на языке C для эффективности используется операция "поразрядное и", `&`, так что их нельзя точно перенести на другие компьютеры, кроме тех, в которых применяется представление с удвоенной точностью целых чисел. Почти все современные компьютеры основаны на арифметике с удвоенной точностью, но для этого алгоритма в действительности операция `&` не нужна. В упр. 10 показано, как получить такую же последовательность чисел на языке FORTRAN, не используя подобные трюки. Несмотря на то что приведенные здесь программы предназначены для генерирования целых чисел с 30 двоичными разрядами, их легко преобразовать в программы генерирования случайных дробей между 0 и 1 с 52 двоичными разрядами на компьютерах, имеющих арифметику с плавающей точкой (см. упр. 11).

Вы, возможно, захотите включить программу *ran_array* в библиотеку программ или найти еще кого-нибудь, кто уже так сделал. Один из способов проверки, реализуются ли программы *ran_array* и *ran_start* в соответствии с операциями, приведенными выше, состоит в запуске следующего элементарного теста программ.

```
main() { register int m; long a[2009];
ran_start(310952);
for (m=0;m<2009;m++) ran_array(a,1009);
printf("%ld\n", ran_x[0]);
ran_start(310952);
for (m=0;m<1009;m++) ran_array(a,2009);
printf("%ld\n", ran_x[0]);
}
```

Должно быть напечатано 461390032 (дважды).

Предостережение: числа, генерируемые программой *ran_array*, не удовлетворяют критерию промежутков между днями рождений, приведенному в разделе 3.3.2J, и обладают другими недостатками, которые иногда возникают в высокоточном моделировании (см. упр. 3.3.2–31 и 3.3.2–35). Один из способов избежать проблем, связанных с критерием промежутков между днями рождения, — просто использовать только половину чисел (пропуская нечетные элементы), но это не панацея от других проблем. Лучшая четная процедура, предложенная Мартином Люшером (Martin Lüscher), обсуждалась в разделе 3.2.2: используйте программу *ran_array* для генерирования, допустим, 1 009 чисел, но применяйте из них только первые 100 (см. упр. 15). *Этот метод теоретически довольно обоснован и не имеет известных недостатков.* Большинство пользователей не нуждаются в таком предостережении, но так, определенно, меньше риска и оно позволяет делать выбор между случайностью и скоростью.

Много известно о линейных конгруэнтных последовательностях, подобных (1), но сравнительно мало исследованы свойства случайных последовательностей

Фибоначчи с запаздыванием, подобных (2). Обе, кажется, похожи на практически надежные, если их использовать с учетом приведенных предостережений.

Когда эта глава была написана в конце 60-х годов, поистине ужасный генератор случайных чисел RANDU использовался в большинстве компьютеров мира (см. раздел 3.3.4). Авторы, внесшие большой вклад в науку о генерировании случайных чисел, часто не знали, что обычных методов их доказательств недостаточно. Особенно заслуживает внимания неприятный пример Алана М. Ферренберга (Alan M. Ferrenberg) и его коллег, опубликованный в *Physical Review Letters* **69** (1992), 3382–3384. Они тестировали свои алгоритмы для трехмерной задачи, рассматривая сначала двумерную задачу с известным ответом, и обнаружили, что предположительно суперкачественные современные генераторы случайных чисел дают неправильные результаты в пятом знаке. А старомодный, как крутящаяся мельница, линейный конгруэнтный генератор $X \leftarrow 16807X \bmod (2^{31} - 1)$ работал прекрасно. Возможно, дополнительные научные исследования покажут, что даже генераторы случайных чисел, рекомендованные здесь, будут неудовлетворительны. Мы надеемся, что этого не случится, но история предупреждает, что нужно быть осмотрительными. Отсюда вытекает наиболее разумная линия поведения — работая с каждой программой метода Монте-Карло, необходимо по крайней мере дважды использовать совершенно различные источники случайных чисел, прежде чем получить решения. Это будет указывать на стабильность результатов, а также оградит от опасного доверия к генераторам со скрытыми недостатками. (Каждый генератор случайных чисел будет "проваливаться" по крайней мере для одной какой-либо задачи.)

Превосходная библиография до 1972 года по генерированию случайного числа составлена Ричардом Нансом и Клодом Оверстритом (мл.) (Richard E. Nance and Claude Overstreet, Jr., *Computing Reviews* **13** (1972), 495–508) и Э. Р. Совьи (E. R. Sowey, *International Stat. Review* **40** (1972), 355–371). Период 1972–1984 годов закрыт Совьи в работах *International Stat. Review* **46** (1978), 89–102; *J. Royal Stat. Soc.* **A149** (1986), 83–107. Последующее развитие обсуждается в книге Шу Тезука (Shu Tezuka, *Uniform Random Numbers* (Boston: Kluwer, 1995)).

Для подробного изучения использования случайных чисел в численном анализе обратитесь к книге J. M. Hammersley and D. C. Handscomb, *Monte Carlo Methods* (London: Methuen, 1964). В ней показано, как некоторые численные методы улучшаются с помощью "квазислучайных" чисел, специально предназначенных для определенных целей (необязательно удовлетворяющие статистическим критериям, которые мы обсуждали). Происхождение метода Монте-Карло для компьютеров обсуждается в работе N. Metropolis and R. Eckhardt in *Stanisław Ulam 1909–1984*, специальный выпуск *Los Alamos Science* **15** (1987), 125–136.

Каждому читателю будет интересно рассмотреть упр. 6, приведенное ниже.

УПРАЖНЕНИЯ

1. [*21*] Используя метод (1), напишите программу со следующими характеристиками для машины MIX.

Вызов последовательности:	`JMP RANDI`
Входные условия:	rA = k — положительное целое число < 5000
Выходные условия:	rA $\leftarrow$ случайное целое число Y, $1 \le Y \le k$, все значения равновероятны; rX =?; переполнение выключено

▸ **2.** [*15*] Кое-кто напуган тем, что компьютеры в будущем будут править миром, но правительство их успокаивает заявлениями, что машина не может сделать ничего нового, так как она только повинуется командам ее создателя, программиста. Леди Лавлейс (Lovelace) в 1844 году писала: "Аналитическая машина не претендует *на создание* чего-либо. Она может сделать все, *если известно, как ей приказать* это выполнить". Ее утверждение развивалось многими философами. Обсудите это высказывание в связи с генераторами случайных чисел.

3. [*32*] (*Игра в кости.*) Напишите программу моделирования бросания двух игральных костей, на каждой из которых выпадают значения 1, 2, ..., 6 с равной вероятностью. Если сумма равна 7 или 11 при первом бросании, то игрок выиграл; если сумма равна 2, 3 или 12, то проиграл. При любой другой сумме называем сумму "очко" и продолжаем бросать игральные кости до тех пор, пока выпадет либо 7 (проигрыш), либо очко (выигрыш).

Играется десять игр. Результат каждого бросания игральных костей можно напечатать в виде $m\,n$, где m и n — цифры, выпавшие на каждой из двух игральных костей, и сопроводить их соответствующими комментариями (например, "змеиные глаза" либо "маленький солдат", либо "трудная дорога" и т. д.).

4. [*40*] (*Солитер, или пасьянс.*) Уйма драгоценного времени уходит на раскладывание пасьянса, и, возможно, автоматика существенно вторгнется в эту область. Напишите программу, которая (i) тасует модель колоды карт; (ii) раскладывает несколько обычных пасьянсов, основанных на порядке карт в колоде, и (iii) выводит на дисплей результат игры, показывая, насколько близко программа подходит к выигрышу. Сохраните собранные статистические данные. Программа по запросу может начать жульничать.

5. [*46*] (*Литературное творчество компьютера.*) Телевизионная программа "Думающая машина" была показана по телесети CBS 26 октября 1960 года. Среди всего прочего она отвела важнейшее место двум небольшим пьесам в стиле "вестерн", написанных компьютером. Здесь помещены два сценария в том виде, в каком они были напечатаны компьютером.

Сценарий #1. (Револьвер в правой руке, деньги в левой руке, стакан на столе, бутылка на столе, кобура на грабителе, револьвер шерифа в правой руке шерифа, кобура шерифа на шерифе.)

ГРАБИТЕЛЬ: (Грабитель за окном.) Идет к двери; открывает дверь; входит; закрывает дверь; идет в угол; кладет в углу деньги; идет к столу; кладет револьвер на стол; сидит и ерзает; сидит за столом; берет стакан правой рукой (пустой); ставит стакан на стол; берет бутылку правой рукой; наливает; ставит бутылку на стол; берет стакан правой рукой; пьет из стакана; ставит стакан на стол; берет бутылку правой рукой; сидит за столом; сидит за столом; идет в угол; идет к окну; идет к столу; ставит бутылку на стол; сидит и ерзает; сидит за столом; сидит и ерзает; идет к окну; идет к столу; подымает стакан правой рукой.

ШЕРИФ: (Шериф за окном.) Видит грабителя (грабитель видит шерифа); идет к двери.

ГРАБИТЕЛЬ: Ставит стакан на стол; берет револьвер правой рукой; проверяет револьвер.

ШЕРИФ: Ждет; открывает двери; видит грабителя (грабитель видит шерифа); входит.

ГРАБИТЕЛЬ: Идет к окну; целится; стреляет; ШЕРИФ ЛЕГКО РАНЕН.

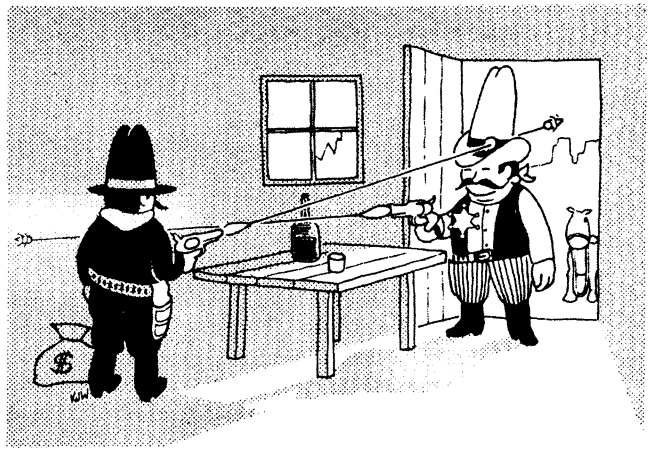
ШЕРИФ: Идет к окну; целится; стреляет; ПРОМАХНУЛСЯ; идет к двери; идет к окну.

ГРАБИТЕЛЬ: Идет к двери; целится; целится.

ШЕРИФ: Целится; стреляет; ПРОМАХНУЛСЯ.

ГРАБИТЕЛЬ: Стреляет; ШЕРИФ ЛЕГКО РАНЕН.

ШЕРИФ: Идет к двери; целится; стреляет; ПРОМАХНУЛСЯ; выходит; целится.



ГРАБИТЕЛЬ: Целится; стреляет; ПРОМАХНУЛСЯ; целится; стреляет; ПРОМАХНУЛСЯ.

ШЕРИФ: Стреляет; ПРОМАХНУЛСЯ; идет к окну; целится; стреляет; ПРОМАХНУЛСЯ.

ГРАБИТЕЛЬ: Целится; стреляет; ПРОМАХНУЛСЯ; целится; стреляет; ПРОМАХНУЛСЯ; целится; стреляет; ШЕРИФ ЛЕГКО РАНЕН.

ШЕРИФ: Целится; стреляет; ПОПАДАЕТ В ГРАБИТЕЛЯ.

ГРАБИТЕЛЬ: Роняет РЕВОЛЬВЕР; грабитель умирает.

ШЕРИФ: Кладет револьвер в кобуру; идет к столу; берет стакан правой рукой (пустой); берет стакан из правой руки в левую руку; берет бутылку правой рукой; наливает; ставит бутылку на стол; берет стакан из левой руки в правую руку; пьет из стакана; берет стакан из правой руки в левую руку; берет бутылку правой рукой; наливает; ставит бутылку на стол; берет стакан из левой руки в правую руку; пьет из стакана; ставит стакан на стол; идет в угол; берет деньги правой рукой; идет к двери; выходит; закрывает дверь. ЗАНАВЕС.

Сценарий #2. (Револьвер в правой руке; деньги в левой руке; стакан на столе; бутылка на столе; кобура на грабителе; револьвер шерифа в правой руке шерифа; кобура шерифа на шерифе.)

ГРАБИТЕЛЬ: (Грабитель за окном.) Идет к двери; открывает дверь; входит; закрывает дверь; идет в угол; кладет деньги в углу; идет к окну; кладет револьвер у окна; прислоняется к окну и смотрит; идет в угол; считает деньги; идет к столу; берет стакан правой рукой (пустой); берет стакан из правой руки в левую руку; берет бутылку правой рукой; наливает; ставит бутылку на стол; берет стакан из левой руки в правую руку; пьет из стакана; ставит стакан на стол; берет бутылку в правую руку; наливает; идет в угол; ставит бутылку в угол; идет к окну; берет револьвер в правую руку; проверяет револьвер; кладет револьвер в кобуру; идет к столу; берет стакан в правую руку; пьет из стакана; идет к окну; ставит стакан у окна.

ШЕРИФ: (Шериф за окном.) Видит грабителя (грабитель видит шерифа); идет к двери.

ГРАБИТЕЛЬ: Берет револьвер из кобуры правой рукой; проверяет револьвер; идет к двери; проверяет револьвер; кладет револьвер у двери.

ШЕРИФ: Открывает двери; видит грабителя (грабитель видит шерифа); входит; идет к окну.

ГРАБИТЕЛЬ: Берет револьвер правой рукой.
ШЕРИФ: Идет к столу.
ГРАБИТЕЛЬ: Целится; стреляет; ПРОМАХНУЛСЯ; прицеливается; стреляет; ПОПАЛ В ШЕРИФА; задул ствол; кладет револьвер в кобуру.
ШЕРИФ: Роняет револьвер; шериф умер.
ГРАБИТЕЛЬ: Идет в угол; берет деньги в правую руку; идет к двери; выходит; закрывает дверь. ЗАНАВЕС.

Внимательный читатель этих сценариев обнаружит представленную здесь в высшей степени напряженную драму. Программа старательно следит за перемещением каждого актера, за тем, что он держит в руках, и т. д. Действия актеров случайные, регулировались определенными вероятностями; вероятности глупых действий возрастали в зависимости от того, как много актер пил и как часто промахивался стрелок. Читатель в состоянии сделать вывод о дополнительных свойствах программы, изучая образцы сценариев. Конечно, даже лучшие сценарии переписывают прежде, чем их поставят, и в особенности когда неопытный писатель готовит начальный набросок. Здесь сценарии точно такие, какие действительно использовались для показа.

Сценарий #1. Музыка.
СП Грабитель всматривается через окно хижины.
КП Лицо грабителя.
СП Грабитель входит в лачугу.
КП Грабитель видит бутылку виски на столе.
КП Шериф за хижиной.
СП Грабитель видит шерифа.
ДП Шериф в дверях за плечом грабителя, оба хватаются за револьверы.
СП Шериф вытаскивает револьвер.
ДП Стреляет. Попадает в грабителя.
СП Шериф поднимает мешки с деньгами.
СП Грабитель шатается.
СП Грабитель умирает. Падает поперек стола после попытки сделать последний выстрел в шерифа.
СП Шериф идет к двери с деньгами.
СП Тело грабителя все еще лежит поперек стола. Камера катится назад. (Смех)

Сценарий #2. Музыка.
КП окно. Появляется грабитель.
СП Грабитель входит в хижину с двумя мешками денег.
СП Грабитель кладет мешки денег на бочку.
КП Грабитель видит виски на столе.
СП Грабитель наливает виски за столом. Идет считать деньги. Смеется.
СП Шериф за хижиной.
СП Вид из окна.
СП Грабитель видит шерифа через окно.
ДП Шериф входит в хижину. Вытаскивает. Стреляет.
КП Шериф. Корчится от выстрела.
СП/2 Шериф идет, шатаясь к столу, чтобы выпить . . . падает мертвый.
СП Грабитель покидает хижину с мешками денег.*

[*Замечание.* КП — "крупный план", СП — "средний план" и т. д. Приведенные сценарии любезно предоставлены Томасом Г. Вулфом (Thomas H. Wolf) — продюсером телепередачи, впервые предложившим идею написания компьютером небольшой пьесы, а также

 Дополнительную информацию можно найти в книге J. E. Pfeiffer, *The Thinking Machine* (New York: J. B. Lippincott, 1962).

Дугласу Т. Россу (Douglas T. Ross) и Гаррисону Р. Морсу (Harrison R. Morse), создавшими программу.]

Летом 1952 года Кристофер Стрейч (Christopher Strachey) использовал техническое обеспечение генератора случайного числа Ferranti Mark I для составления следующего письма.

> Милая голубушка
>
> Моя полная сочувствия привязанность прелестно привлекает ваш показной энтузиазм. Вы — мое нежное обожание, мое затаенное обожание. Мои приятельские чувства, затаив дыхание, надеются на ваш дорогой пыл. Мое томящееся от любви поклонение лелеет ваше жадное рвение.
>
> Ваш грустящий
> М. У. К.

[*Encounter* **3** (1954), 4, 25–31; другой пример появился в статье на Electronic Computers в 64 издании *Pears Cyclopedia* (London, 1955), 190–191.]

У читателя, несомненно, есть множество идей, как научить компьютер литературному творчеству. Это и является целью данного упражнения.

▶ **6.** [*40*] Просмотрите библиотеку программ каждого установленного в вашей организации компьютера и замените плохие генераторы случайных чисел хорошими. Попробуйте избежать сильных потрясений от того, что вы обнаружите.

▶ **7.** [*M40*] Решительный программист формирует свои файлы, используя линейную конгруэнтную последовательность $\langle X_n \rangle$ с периодом 2^{32}, которая генерируется (1) для $m = 2^{32}$. Он берет наибольшие значащие двоичные разряды $\lfloor X_n/2^{16} \rfloor$ и добавляет их в свои данные с исключающим "или", но держит в секрете параметры a, c и X_0.

Покажите, что это не очень надежная схема, и придумайте метод нахождения множителя a и первой разности $X_1 - X_0$ за разумное приемлемое время, задав только значения $\lfloor X_n/2^{16} \rfloor$ для $0 \le n < 150$.

8. [*M15*] Предложите хороший метод выполнения проверки, хорошо ли работают линейные конгруэнтные генераторы.

9. [*HM32*] Пусть $X_0, X_1, \ldots$ — числа, полученные программой *ran_array* после того, как *ran_start* инициализирует процесс генерирования с начальным значением s. Рассмотрим полиномы

$$P_n(z) = X_{n+62}z^{99} + X_{n+61}z^{98} + \cdots + X_n z^{37} + X_{n+99}z^{36} + \cdots + X_{n+64}z + X_{n+63}.$$

a) Докажите, что $P_n(z) \equiv z^{h(s)-n}$ (по модулю 2 и $z^{100} + z^{37} + 1$) для некоторого показателя $h(s)$.

b) Выразите $h(s)$ в терминах двоичного представления s.

c) Докажите, что если $X'_0, X'_1, \ldots$ — последовательность чисел, полученных той же программой при начальном значении $s' \neq s$, то $X_{n+k} \equiv X'_{n'+k}$ (по модулю 2) для $0 \le k < 100$ только тогда, когда $|n - n'| \ge 2^{70} - 1$.

10. [*22*] Преобразуйте программы *ran_array* и *ran_start* на языке C в программы на языке FORTRAN 77 генерирования той же последовательности чисел.

▶ **11.** [*M25*] Предположим, что арифметика с плавающей точкой на числах, имеющих тип `double`, правильно выполняет округление в смысле раздела 4.2.2 (т. е. точно тогда, когда значения должным образом ограничены). Преобразуйте программы *ran_array* и *ran_start* на языке C в сходные программы, которые выдают случайные дроби с двойной точностью в $[0\,..\,1)$ вместо целых чисел с 30-ю двоичными разрядами.

▶ **12.** [*M21*] Какой генератор случайных чисел будет подходящим для мини-компьютера, который имеет арифметику только для целых чисел в области $[-32768\,..\,32767]$?

13. [*M25*] Сравните генератор вычитания с заимствованием из упр. 3.2.1.1–12 с генератором Фибоначчи с запаздыванием, реализованным в программах этого раздела.

▸ **14.** [*M35*] (*Будущее против прошлого.*) Пусть $X_n = (X_{n-37} + X_{n-100}) \bmod 2$. Рассмотрим последовательность

$$\langle Y_0, Y_1, \ldots \rangle = \langle X_0, X_1, \ldots, X_{99}, X_{200}, X_{201}, \ldots, X_{299}, X_{400}, X_{401}, \ldots, X_{499}, X_{600}, \ldots \rangle.$$

(Она соответствует неоднократно вызываемой программе *ran_array*(a,200), когда рассматриваются только младшие значащие двоичные разряды после отбрасывания половины элементов.) Следующий эксперимент был повторен один миллион раз с использованием последовательности $\langle Y_n \rangle$: "Генерируем 100 случайных двоичных разрядов, затем, если 60 или более из них равны нулю, генерируем еще один и печатаем его". В результате было напечатано 14 527 нулей и 13 955 единиц, но вероятность того, что 28 482 случайных двоичных разряда содержат самое большее 13 955 единиц, равна приблизительно .000358.

Дайте математическое объяснение, почему так много нулей будет на выходе.

▸ **15.** [*25*] Напишите на языке C программу генерирования для случайных целых чисел, которые получаются с помощью программы *ran_array*, отбрасывая все, кроме первых 100, элементы из каждых 1 009 элементов, как рекомендуется в разделе.