Генерация (псевдо) случайных чисел в языке СИ

Случайные числа. Генераторы случайных чисел

Современные компиляторы обладают собственной реализацией генератора псевдослучайных последовательностей. Основная сложность генерации последовательности псевдослучайных чисел на компьютере в том, что компьютеры детерминированы по своей сути. Компьютер может находиться только в конечном количестве состояний (количество состояний огромно, но все-таки конечно). Следовательно любой датчик случайных чисел по определению периодичен. Все периодическое –

Генераторы случайных чисел

С криптографической точки зрения они являются непригодными.
Лучшее, что может произвести компьютер – это псевдослучайная последовательность.
Период такой последовательности должен быть таким, чтобы конечная последовательность разумной длины не была периодической.
Относительно короткие непериодические подпоследовательности должны быть как можно более неотличимы от случайных последовательностей, в частности, соответствовать различным критериям случайности.

Генераторы случайных чисел

Генератор последовательности называется случайным, если он не может быть достоверно воспроизведен, т.е. дважды запуская генератор с абсолютно одинаковыми исходными данными (по крайней мере, на пределе возможностей), мы получим случайные различные последовательности.

Генератор псевдослучайной последовательности выдает поток битов, который выглядит случайными, но в действительности является детерминированным и может быть в точности воспроизведен.

Линейный конгруэнтный генератор псевдослучайных чисел

- **Линейный конгруэнтный генератор (ЛКГ) – это последовательность чисел от 0 до** *m −1, удовлетворяющая следующему рекуррентному выражению* $X_{k+1} = X_k a + b \bmod m$, X_0 *начальное значение, a – множитель, b – приращение, m – модуль. У такого* генератора период меньше *m . Если a , b , m правильно выбраны, то генератор* является генератором максимальной длины и имеет период *m.*

Линейный конгруэнтный генератор псевдослучайных чисел

Если инкремент *b равен нулю, то есть генератор имеет вид:*

$$X_{k+1} = X_k a \bmod m,$$

получим самую простую последовательность, которую можно предложить для генератора с равномерным распределением. При соответствующем выборе констант *a* может принимать значения 75 либо 16 807 и *m принимать* $2^{31}-1$ *ия* 2 147 483 647 получим генератор с максимальным периодом повторения. Эти константы были предложены учеными Парком и Миллером, поэтому генератор вида:

$$X_{k+1} = 75X_k \bmod(2^{31}-1),$$

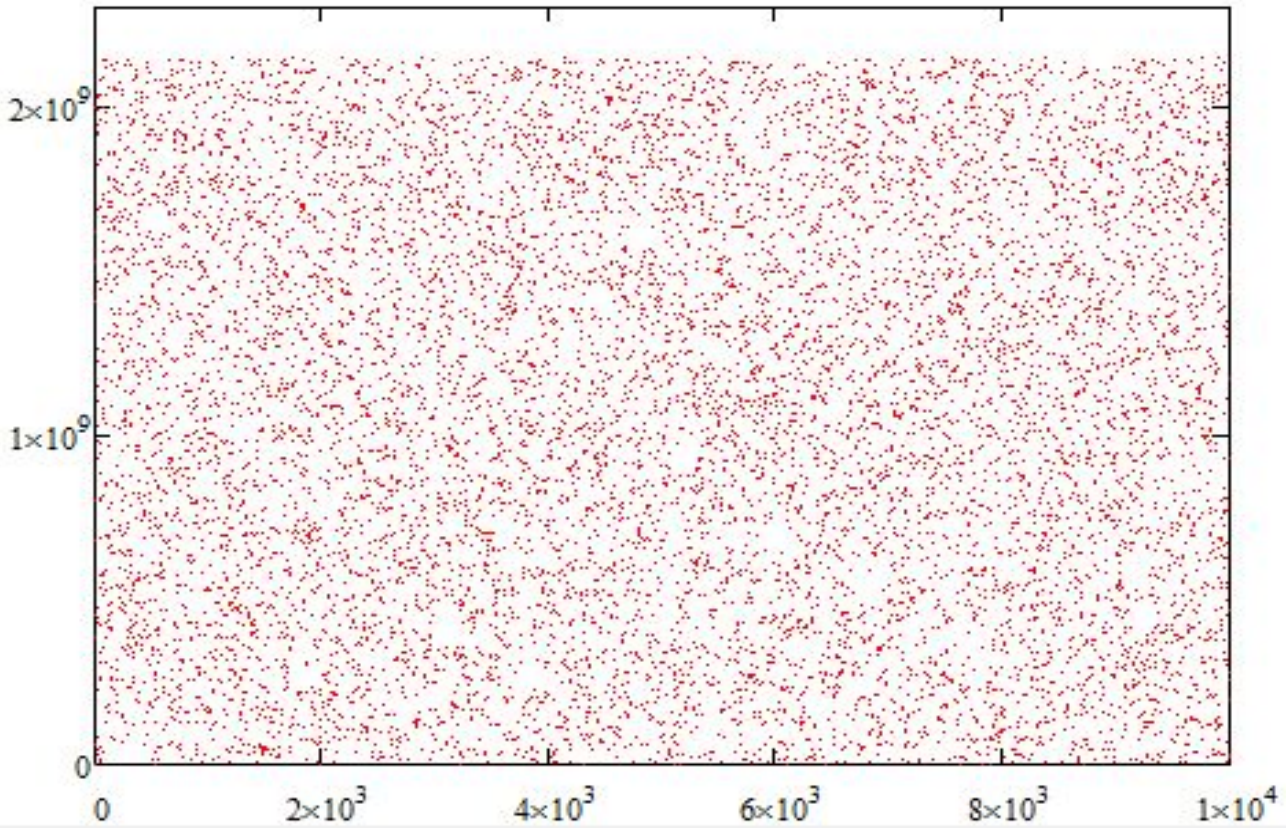
называется ***генератором Парка-Миллера.***

Линейный конгруэнтный генератор псевдослучайных чисел

Реализация генератора Парка-Миллера.

$k := 0 .. 10000 \qquad x_0 := 2$

$x_{k+1} := \mathrm{mod}\left(75 \cdot x_k, 2^{31} - 1\right) \qquad 2^{31} - 1 = 2147483647$



Псевдослучайные числа в СИ

Функция, генерирующая псевдослучайные числа, имеет прототип в файле библиотеки stdlib.h

В этой библиотеке содержится функция rand(), которая и генерирует случайное число. Эта функция не принимает никакие параметры.

Функция rand() возвращает целое число от 0 до значения присвоенного константе RAND_MAX. Значение RAND_MAX зависит от системы и определено в заголовочном файле stdlib.h. Так, например, оно может быть равно 32767 (двухбайтовое целое) или 2147483647 (четырехбайтовое целое).

Псевдослучайные числа в СИ

Функция, генерирующая псевдослучайные числа, имеет прототип в файле библиотеки stdlib.h

В этой библиотеке содержится функция rand(), которая и генерирует случайное число. Эта функция не принимает никакие параметры.

Функция rand() возвращает целое число от 0 до значения присвоенного константе RAND_MAX. Значение RAND_MAX зависит от системы и определено в заголовочном файле stdlib.h. Так, например, оно может быть равно 32767 (двухбайтовое целое) или 2147483647 (четырехбайтовое целое).

Псевдослучайные числа в СИ

Значение RAND_MAX равно 32767 (двухбайтовое целое).

```c
#include <stdio.h>
#include <stdlib.h>

int main ()
{
      printf("max= %d\n", RAND_MAX);
      return 0;   }
```

```
max= 32767

Process returned 0 (0x0)   execution time : 1.043 s
Press any key to continue.
```

Псевдослучайные числа в СИ

Код ниже выводит на экран 25 случайных чисел. В теле цикла осуществляется переход на новую строку после каждых выведенных на экран пяти чисел.

```c
#include <stdio.h>
#include <stdlib.h>

int main () {  int i;

     for (i = 1; i <= 25; i++) {
          printf("%10d", rand());
          if (i % 5 == 0) printf("\n");
     }
     return 0;  }
```

```
        41      18467       6334      26500      19169
     15724      11478      29358      26962      24464
      5705      28145      23281      16827       9961
       491       2995      11942       4827       5436
     32391      14604       3902        153        292

Process returned 0 (0x0)   execution time : 0.849 s
Press any key to continue.
```

Псевдослучайные числа в СИ

Для этого используется выражение, в котором находится остаток от деления i на 5, результат сравнивается с 0. Чтобы после первого числа не происходил переход на новую строку, i сначала присваивается единица, а не ноль (т.к. 0 делится на 5 без остатка).

```c
#include <stdio.h>
#include <stdlib.h>

int main () {   int i;

     for (i = 1; i <= 25; i++) {
          printf("%10d", rand());
          if (i % 5 == 0) printf("\n");
     }
     return 0;   }
```

```
        41      18467       6334      26500      19169
     15724      11478      29358      26962      24464
      5705      28145      23281      16827       9961
       491       2995      11942       4827       5436
     32391      14604       3902        153        292

Process returned 0 (0x0)   execution time : 0.849 s
Press any key to continue.
```

Функция rand

При каждом запуске программы числа остаются одинаковыми. Даже если перекомпилировать программу, результат не изменится. Данный эффект связан с тем, что начальное (инициализирующее) число, которое подставляется в формулу вычисления первого и последующих псевдослучайных чисел, для каждой системы всегда одно и то же.

Функция srand

Однако это начальное число можно изменить с помощью функции **srand(),** которой в качестве параметра передается любое целое число. Если вы зададите конкретный аргумент для функции, например, **srand(1000),** то от вызова к вызову программы числа будут также одни и те же. Хотя и не те, что были бы без srand(). Поэтому появляется проблема, как сделать так, чтобы аргумент для srand() был тоже случайным?

Функция srand

- Пользователь программы сам может задавать инициализирующее значение. Но чаще всего это не является полноценным выходом из ситуации. Поэтому инициализирующее значение привязывают к какому-либо процессу, протекающему в операционной системе, например, к часам. Время (учитывая не только время суток, но и дату) никогда не бывает одинаковым. Значит значение для srand(), преобразованное в целое из системного времени, будет различным.

Функция srand

- Текущее время можно узнать с помощью функции time(), прототип которой описан в файле **time.h.**
- Функция time в соответствии с системными часами возвращает количество секунд, прошедших от 00:00:00 значения времени по Гринвичу,т.е. с 1 января 1980 года (или 1970 года). Возвращаемое значение хранится в расположении, заданном по timeptr. Если возвращаемое значение не запомнилось, адрес timeptr является **NULL**.
- Передав time() в качестве параметра NULL, мы получим целое число, которое можно передать

Функция srand

Код ниже выводит на экран 10 случайных чисел

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

int main()
{
 int n,i;
    n=10;
    srand(time (NULL));
  int a [n];
  for(i=0;i<n;i++)
  { a[i]=rand() ;  }

   for(i=0;i<n;i++)
  {  printf(" a[%d] =%d\n",i,a[i]); }
  printf(" year =%d \n" ,time (NULL)/3600/24/365);
                return 0;
```

```
a[0] =14892
a[1] =1655
a[2] =22919
a[3] =12147
a[4] =7047
a[5] =15410
a[6] =24623
a[7] =1065
a[8] =17048
a[9] =14756
year =48

Process returned 0 (0x0)   execution time : 0.062 s
Press any key to continue.
```

Функция rand

Функция возвращает случайное целое число в диапазоне от нуля до RAND_MAX. Чтобы ограничить сверху случайные числа, можно воспользоваться операцией получения остатка от деления. Остаток от деления на числа K всегда меньше числа K. Например, при делении на 4 могут получиться остатки 0, 1, 2 и 3. Поэтому если вы хотите ограничить сверху случайные числа числом K, то можно взять остаток от деления на K.

Функция rand

Генерация пяти случайных целых чисел меньших K=

```c
#include <stdio.h>
#include <stdlib.h>

int main( ) { int K = 10;
/* генерируем пять случайных целых чисел меньших K=10 */
  printf("%d\n", rand()%K);
  printf("%d\n", rand()%K);
  printf("%d\n", rand()%K);
  printf("%d\n", rand()%K);
  printf("%d\n", rand()%K);
  return 0;
}
```

```
1
7
4
0
9

Process returned 0 (0x0)   execution time : 0.063 s
Press any key to continue.
```

Функция rand

В общем случае если нам нужно получить числа из отрезка [A;B], то можно воспользоваться следующей конструкцией:

A + rand()%(B-A+1).

Например, необходимо получить случайные числа от 80 до 100.

Для этого используем выражение

80 + rand()%(100 - 80 + 1)

Функция rand

Пример программы генерирующей числа в диапазоне [80;100]

```
#include <stdio.h>
#include <stdlib.h>

int main( ) {
 int n,i;
    n=10;
    int a [n];
  for(i=0;i<n;i++)
  { a[i]=80 + rand()%(100 - 80 + 1) ; }
   for(i=0;i<n;i++)
  {  printf(" a[%d] =%d \n",i,a[i]); }
   return 0;
}
```

```
a[0] =100
a[1] =88
a[2] =93
a[3] =99
a[4] =97
a[5] =96
a[6] =92
a[7] =80
a[8] =99
a[9] =100
```

Функция rand

- Для получения псевдослучайных вещественных значений в заданном диапазоне удобно использовать следующую формулу:

```
(float) rand() / RAND_MAX * (max - min) + min
```

- В этом выражении целое значение, возвращаемое функцией rand() явным образом преобразуется в вещественное, т.к. в противном случае всегда будет получаться нулевое значение

Функция rand

Пример программы генерирующей числа в диапазоне [0;1]

```c
#include <stdio.h>
#include <stdlib.h>

int main( ) {
 int n,i; float min, max;
    n=10; min = 0; max = 1;
    float a [n];
  for(i=0;i<n;i++)
  { a[i]=min + (float)rand()/RAND_MAX*(max - min);
     }
   for(i=0;i<n;i++)
  {  printf(" a[%d] =%f\n",i,a[i]); }
   return 0;
}
```

```
a[0] =0.001251
a[1] =0.563585
a[2] =0.193304
a[3] =0.808740
a[4] =0.585009
a[5] =0.479873
a[6] =0.350291
a[7] =0.895962
a[8] =0.822840
a[9] =0.746605
```

Комментарии к практической работе 4

4. ВЫЧИСЛЕНИЕ КОНЕЧНЫХ СУММ

Часто для вычисления значений некоторой функции используют разложение этой функции в бесконечный ряд. Так, например, функцию $y = e^x$ можно представить в виде ряда Маклорена:

$$e^x = 1 + x + \frac{x^2}{2!} + \ldots + \frac{x^n}{n!} + R_n,$$

где $\frac{x^n}{n!}$ – общий член ряда, а R_n – остаточный член ряда. Так как ряд сходится, т.е. остаточный член при увеличении значения n стремится к нулю, то функцию $y = e^x$ можно аппроксимировать конечной суммой вида:

$$S = 1 + x + \frac{x^2}{2!} + \ldots + \frac{x^n}{n!} \approx e^x, \quad \text{если } R_n \le \varepsilon,$$

где ε – заранее заданная точность аппроксимации.

Комментарии к практической работе 4

4.1. Задания на нахождение конечной суммы ряда

а) Решить задачу вычисления конечной суммы ряда для заданного числа n при изменении аргумента x в заданном диапазоне, количество шагов изменения x равно 10,

б) Определить число членов конечной суммы n, позволяющее аппроксимировать функцию с заданной точностью ε (например, $\varepsilon = 10^{-5}$), исходя из критерия $|S_n - S_{n-1}| = |a_n| < \varepsilon$.

№	Сумма	Диапазон изменения аргумента	n	Функция $y(x)$
1	$S = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots + (-1)^n \frac{x^{2n+1}}{(2n+1)!}$	$0{,}1 \div 1$	10	$\sin(x)$