Linear congruential generator

From Rosetta Code

Linear congruential generator
You are encouraged to solve this task according to the task description, using any language you may know.

The linear congruential generator is a very simple example of a random number generator. All linear congruential generators use this formula:

- $r_{n+1} = a \times r_n + c \pmod m$

Where:

- r_0 is a seed.
- r_1, r_2, r_3, ..., are the random numbers.
- a, c, m are constants.

If one chooses the values of a, c and m with care, then the generator produces a uniform distribution of integers from 0 to $m - 1$.

LCG numbers have poor quality. r_n and r_{n+1} are not independent, as true random numbers would be. Anyone who knows r_n can predict r_{n+1}, therefore LCG is not cryptographically secure. The LCG is still good enough for simple tasks like Miller-Rabin primality test, or FreeCell deals. Among the benefits of the LCG, one can easily reproduce a sequence of numbers, from the same r_0. One can also reproduce such sequence with a different programming language, because the formula is so simple.

The task is to replicate two historic random number generators. One is the `rand()` function from BSD libc, and the other is the `rand()` function from the Microsoft C Runtime (MSCVRT.DLL). Each replica must yield the same sequence of integers as the original generator, when starting from the same seed.

In these formulas, the seed becomes $state_0$. The random sequence is $rand_1$, $rand_2$ and so on.

BSD formula:

- $state_{n+1} = 1103515245 \times state_n + 12345 \pmod{2^{31}}$
- $rand_n = state_n$
- $rand_n$ is in range 0 to 2147483647.

Microsoft formula:

- $state_{n+1} = 214013 \times state_n + 2531011 \pmod{2^{31}}$
- $rand_n = state_n \div 2^{16}$
- $rand_n$ is in range 0 to 32767.

The BSD formula was so awful that FreeBSD switched to a different formula. More info is at Random number generator (included)#C.

Contents

Ada

We first specify a generic package LCG:

```
generic
   type Base_Type is mod <>;
   Multiplyer, Adder: Base_Type;
   Output_Divisor: Base_Type := 1;
package LCG is

   procedure Initialize(Seed: Base_Type);
   function Random return Base_Type;
   -- changes the state and outputs the result

end LCG;
```

Then we provide a generic implementation:

```
package body LCG is

   State: Base_Type := Base_Type'First;

   procedure Initialize(Seed: Base_Type) is
   begin
      State := Seed;
   end Initialize;

   function Random return Base_Type is
   begin
      State := State * Multiplyer + Adder;
      return State / Output_Divisor;
   end Random;

end LCG;
```

Next, we define the MS- and BSD-instantiations of the generic package:

```
with Ada.Text_IO, LCG;

procedure Run_LCGs is

   type M31 is mod 2**31;
```

```
   package BSD_Rand is new LCG(Base_Type => M31, Multiplyer => 1103515245,
                               Adder => 12345);

   package MS_Rand  is new LCG(Base_Type => M31, Multiplyer => 214013,
                               Adder => 2531011, Output_Divisor => 2**16);

begin
   for I in 1 .. 10 loop
      Ada.Text_IO.Put_Line(M31'Image(BSD_Rand.Random));
   end loop;
   for I in 1 .. 10 loop
       Ada.Text_IO.Put_Line(M31'Image(MS_Rand.Random));
   end loop;
end Run_LCGs;
```

Finally, we run the program, which generates the following output (note that the first ten lines are from the BSD generator, the next ten from the MS generator):

```
 12345
 1406932606
 654583775
 1449466924
 229283573
 1109335178
 1051550459
 1293799192
 794471793
 551188310
 38
 7719
 21238
 2437
 8855
 11797
 8365
 32285
 10450
 30612
```

AutoHotkey

```
a := 0, b:= [0]
Loop, 10
        BSD .= "`t" (a :=  BSD(a)) "`n"
,       b := MS(b[1])
,       MS .= "`t" (b[2]) "`n"

MsgBox, % "BSD:`n" BSD "`nMS:`n" MS

BSD(Seed) {
        return, Mod(1103515245 * Seed + 12345, 2147483648)
}

MS(Seed) {
        Seed := Mod(214013 * Seed + 2531011, 2147483648)
        return, [Seed, Seed // 65536]
}
```

Output:

```
BSD:
        12345
        1406932606
        654583775
        1449466924
        229283573
        1109335178
        1051550459
        1293799192
        794471793
        551188310

MS:
        38
        7719
        21238
        2437
        8855
        11797
        8365
        32285
        10450
        30612
```

BBC BASIC

Works with: BBC BASIC for Windows

```
      @% = &D0D
      PRINT "MS generator:"
      dummy% = FNrandMS(0)
      FOR i% = 1 TO 10
        PRINT FNrandMS(-1)
      NEXT
      PRINT '"BSD generator:"
      dummy% = FNrandBSD(0)
      FOR i% = 1 TO 10
        PRINT FNrandBSD(-1)
      NEXT
      END

      DEF FNrandMS(seed%)
      PRIVATE state%
      IF seed% >= 0 THEN
        state% = seed%
      ELSE
        state% = FNmuladd(state%, 214013, 2531011)
      ENDIF
      = state% >> 16

      DEF FNrandBSD(seed%)
      PRIVATE state%
      IF seed% >= 0 THEN
        state% = seed%
      ELSE
        state% = FNmuladd(state%, 1103515245, 12345)
      ENDIF
```

```
      = state%

      DEF FNmuladd(A%,B%,C%) : PRIVATE M% : LOCAL P% : IF M% = 0 DIM P% 8
      IF P% THEN [OPT 0 : .M% mul ebx : add eax,ecx : btr eax,31 : ret :]
      = USR M%
```

Output:

```
MS generator:
           38
         7719
        21238
         2437
         8855
        11797
         8365
        32285
        10450
        30612

BSD generator:
        12345
   1406932606
    654583775
   1449466924
    229283573
   1109335178
   1051550459
   1293799192
    794471793
    551188310
```

Headline text

bc

Translation of: dc
Works with: GNU bc
Works with: OpenBSD bc

As with dc, bc has no bitwise operators.

```
/* BSD rand */

define rand() {
        randseed = (randseed * 1103515245 + 12345) % 2147483648
        return randseed
}

randseed = 1
rand(); rand(); rand(); print "\n"

/* Microsoft rand */
```

```
define rand() {
        randseed = (randseed * 214013 + 2531011) % 2147483648
        return randseed / 65536
}

randseed = 1
rand(); rand(); rand(); print "\n"
```

Bracmat

```
( 2^31:?RANDMAX
& 2^-16:?rshift
& (randBSD=mod$(!seed*1103515245+12345.!RANDMAX):?seed)
& ( randMS
  =   div
    $ ((mod$(!seed*214013+2531011.!RANDMAX):?seed)*!rshift.1)
  )
& out$\nBSD
& 0:?seed
& 0:?i
& whl'(1+!i:~>10:?i&out$!randBSD)
& out$\nMicrosoft
& 0:?seed
& 0:?i
& whl'(1+!i:~>10:?i&out$!randMS)
)
```

Output:

```
BSD
12345
1406932606
654583775
1449466924
229283573
1109335178
1051550459
1293799192
794471793
551188310

Microsoft
38
7719
21238
2437
8855
11797
8365
32285
10450
30612
```

C

In a pretended lib style, this code produces a rand() function depends on compiler

macro: `gcc -DMS_RAND` uses MS style, otherwise it's BSD rand by default.

```
#include <stdio.h>

/* always assuming int is at least 32 bits */
int rand();
int rseed = 0;

inline void srand(int x)
{
        rseed = x;
}

#ifndef MS_RAND
#define RAND_MAX ((1U << 31) - 1)

inline int rand()
{
        return rseed = (rseed * 1103515245 + 12345) & RAND_MAX;
}

#else /* MS rand */

#define RAND_MAX_32 ((1U << 31) - 1)
#define RAND_MAX ((1U << 15) - 1)

inline int rand()
{
        return (rseed = (rseed * 214013 + 2531011) & RAND_MAX_32) >> 16;
}

#endif/* MS_RAND */

int main()
{
        int i;
        printf("rand max is %d\n", RAND_MAX);

        for (i = 0; i < 100; i++)
                printf("%d\n", rand());

        return 0;
}
```

C++

```
#include <iostream>

//--------------------------------------------------------------------------------------------
using namespace std;

//--------------------------------------------------------------------------------------------
class mRND
{
public:
    void seed( unsigned int s ) { _seed = s; }

protected:
```

```
    mRND() : _seed( 0 ), _a( 0 ), _c( 0 ), _m( 2147483648 ) {}
    int rnd() { return( _seed = ( _a * _seed + _c ) % _m ); }

    int _a, _c;
    unsigned int _m, _seed;
};
//---------------------------------------------------------------------------------------------------
class MS_RND : public mRND
{
public:
    MS_RND()  { _a = 214013; _c = 2531011; }
    int rnd() { return mRND::rnd() >> 16; }
};
//---------------------------------------------------------------------------------------------------
class BSD_RND : public mRND
{
public:
    BSD_RND() { _a = 1103515245; _c = 12345; }
    int rnd() { return mRND::rnd(); }
};
//---------------------------------------------------------------------------------------------------
int main( int argc, char* argv[] )
{
    BSD_RND bsd_rnd;
    MS_RND ms_rnd;

    cout << "MS RAND:" << endl << "========" << endl;
    for( int x = 0; x < 10; x++ )
        cout << ms_rnd.rnd() << endl;

    cout << endl  << "BSD RAND:" << endl << "=========" << endl;
    for( int x = 0; x < 10; x++ )
        cout << bsd_rnd.rnd() << endl;

    cout << endl << endl;
    system( "pause" );
    return 0;
}
//---------------------------------------------------------------------------------------------------
```

Output:

```
MS RAND:
========
38
7719
21238
2437
8855
11797
8365
32285
10450
30612

BSD RAND:
=========
12345
1406932606
654583775
1449466924
```

```
229283573
1109335178
1051550459
1293799192
794471793
551188310
```

Clojure

```
(defn iterator [a b]
  (fn[x] (mod (+ (* a x) b) (bit-shift-left 1 31))))

(def bsd (drop 1 (iterate (iterator 1103515245 12345) 0)))

(def ms (drop 1 (for [x (iterate  (iterator 214013 2531011) 0)] (bit-shift-right x 16))))

(take 10 bsd) ;-> (12345 1406932606 654583775 1449466924 229283573 1109335178 1051550459 1293799192 7944
(take 10 ms) ;-> (38 7719 21238 2437 8855 11797 8365 32285 10450 30612)
```

Common Lisp

```
(defun make-rng (&key (seed 0) (mode nil))
  "returns an RNG according to :seed and :mode keywords
  default mode: bsd
  default seed: 0 (should be 1 actually)"
  (if (eql mode 'ms)
    #'(lambda ()
        (ash (setf seed (mod (+ (* 214013 seed) 2531011) (expt 2 31))) -16))
    #'(lambda () (setf seed (mod (+ (* seed 1103515245) 12345) (expt 2 31))))))

(let ((rng (make-rng)))
     (dotimes (x 10) (format t "BSD: ~d~%" (funcall rng))))

(let ((rng (make-rng :mode 'ms :seed 1)))
     (dotimes (x 10) (format t "MS: ~d~%" (funcall rng))))
```

C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FreeCellDeals
{
    public class LCG
    {
        private int _state;
```

```
        public bool Microsoft { get; set;}
        public bool BSD
        {
            get
            {
                return !Microsoft;
            }
            set
            {
                Microsoft = !value;
            }
        }

        public LCG(bool microsoft = true)
        {
            _state = (int)DateTime.Now.Ticks;
            Microsoft = microsoft;
        }

        public LCG(int n, bool microsoft = true)
        {
            _state = n;
            Microsoft = microsoft;
        }

        public int Next()
        {
            if (BSD)
            {
                return _state = (1103515245 * _state + 12345) & int.MaxValue;
            }
            return ((_state = 214013 * _state + 2531011) & int.MaxValue) >> 16;
        }

        public IEnumerable<int> Seq()
        {
            while (true)
            {
                yield return Next();
            }
        }
    }

    class Program
    {
        static void Main()
        {
            LCG ms = new LCG(0, true);
            LCG bsd = new LCG(0,false);
            Console.WriteLine("Microsoft");
            ms.Seq().Take(10).ToList().ForEach(Console.WriteLine);
            Console.WriteLine("\nBSD");
            bsd.Seq().Take(10).ToList().ForEach(Console.WriteLine);
            Console.ReadKey();
        }
    }
}
```

Output:

```
Microsoft
```

```
38
7719
21238
2437
8855
11797
8365
32285
10450
30612

BSD
12345
1406932606
654583775
1449466924
229283573
1109335178
1051550459
1293799192
794471793
551188310
```

D

```
struct LinearCongruentialGenerator {
    enum uint RAND_MAX = (1U << 31) - 1;
    uint seed = 0;

    uint randBSD() pure nothrow @nogc {
        seed = (seed * 1_103_515_245 + 12_345) & RAND_MAX;
        return seed;
    }

    uint randMS() pure nothrow @nogc {
        seed = (seed * 214_013 + 2_531_011) & RAND_MAX;
        return seed >> 16;
    }
}

void main() {
    import std.stdio;

    LinearCongruentialGenerator rnd;

    foreach (immutable i; 0 .. 10)
        writeln(rnd.randBSD);
    writeln;

    rnd.seed = 0;
    foreach (immutable i; 0 .. 10)
        writeln(rnd.randMS);
}
```

Output:

```
12345
1406932606
```

```
654583775
1449466924
229283573
1109335178
1051550459
1293799192
794471793
551188310

38
7719
21238
2437
8855
11797
8365
32285
10450
30612
```

dc

dc has no bitwise operations, so this program uses the modulus operator (`2147483648 %`) and division (`65536 /`). Fortunately, *dc* numbers cannot overflow to negative, so the modulus calculation involves only non-negative integers.

For BSD rand():

```
[*
 * lrx -- (random number from 0 to 2147483647)
 *
 * Returns a number from the BSD rand() sequence.
 * Seeded by storing a seed in register R.
 *]sz
[lR 1103515245 * 12345 + 2147483648 % d sR]sr

[* Set seed to 1, then print the first 3 random numbers. *]sz
1 sR
lrx psz lrx psz lrx psz
```

```
1103527590
377401575
662824084
```

For Microsoft rand():

```
[*
 * lrx -- (random number from 0 to 32767)
 *
 * Returns a number from the Microsoft rand() sequence.
 * Seeded by storing a seed in register R.
 *]sz
[lR 214013 * 2531011 + 2147483648 % d sR 65536 /]sr

[* Set seed to 1, then print the first 3 random numbers. *]sz
1 sR
```

```
lrx psz lrx psz lrx psz
```

```
41
18467
6334
```

F#

```
module lcg =
    let bsd seed =
        let state = ref seed
        (fun (_:unit) ->
            state := (1103515245 * !state + 12345) &&& System.Int32.MaxValue
            !state)

    let ms seed =
        let state = ref seed
        (fun (_:unit) ->
            state := (214013 * !state + 2531011) &&& System.Int32.MaxValue
            !state / (1<<<16))
```

```
let rndBSD = lcg.bsd 0;;
let BSD=[for n in [0 .. 9] -> rndBSD()];;

let rndMS = lcg.ms 0;;
let MS=[for n in [0 .. 9] -> rndMS()];;

val BSD : int list =
  [12345; 1406932606; 654583775; 1449466924; 229283573; 1109335178; 1051550459;
   1293799192; 794471793; 551188310]
val MS : int list =
  [38; 7719; 21238; 2437; 8855; 11797; 8365; 32285; 10450; 30612]
```

Forth

```
1 31 lshift 1- constant MAX-RAND-BSD
1 15 lshift 1- constant MAX-RAND-MS

variable seed                           \ seed variable

: (random) seed @ * + dup seed ! ;    ( -- n)
: BSDrandom MAX-RAND-BSD 12345 1103515245 (random) and ;
: MSrandom MAX-RAND-MS 2531011 214013 (random) 16 rshift and ;

: test-random
  1 seed ! cr ." BSD (seed=1)" cr
  5 0 do BSDrandom . cr loop
  1 seed ! cr ." MS  (seed=1)" cr
  5 0 do MSrandom . cr loop
;

test-random
```

Output:

```
BSD (seed=1)
1103527590
377401575
662824084
1147902781
2035015474

MS  (seed=1)
41
18467
6334
26500
19169
```

Fortran

Works with: Fortran version 90 and later

```
module lcgs
  implicit none

  integer, parameter :: i64 = selected_int_kind(18)
  integer, parameter :: a1 = 1103515245, a2 = 214013
  integer, parameter :: c1 = 12345, c2 = 2531011
  integer, parameter :: div = 65536
  integer(i64), parameter :: m = 2147483648_i64  ! need to go to 64 bits because
                                                 ! of the use of signed integers
contains

function bsdrand(seed)
  integer :: bsdrand
  integer, optional, intent(in) :: seed
  integer(i64) :: x = 0

  if(present(seed)) x = seed
  x = mod(a1 * x + c1, m)
  bsdrand = x
end function

function msrand(seed)
  integer :: msrand
  integer, optional, intent(in) :: seed
  integer(i64) :: x = 0

  if(present(seed)) x = seed
  x = mod(a2 * x + c2, m)
  msrand = x / div
end function
end module

program lcgtest
  use lcgs
  implicit none
  integer :: i

  write(*, "(a)") "      BSD           MS"
  do i = 1, 10
```

```
    write(*, "(2i12)") bsdrand(), msrand()
  end do
end program
```

Output

```
     BSD           MS
       12345          38
  1406932606        7719
   654583775       21238
  1449466924        2437
   229283573        8855
  1109335178       11797
  1051550459        8365
  1293799192       32285
   794471793       10450
   551188310       30612
```

Go

```
package main

import "fmt"

// basic linear congruential generator
func lcg(a, c, m, seed uint32) func() uint32 {
    r := seed
    return func() uint32 {
        r = (a*r + c) % m
        return r
    }
}

// microsoft generator has extra division step
func msg(seed uint32) func() uint32 {
    g := lcg(214013, 2531011, 1<<31, seed)
    return func() uint32 {
        return g() / (1 << 16)
    }
}

func example(seed uint32) {
    fmt.Printf("\nWith seed = %d\n", seed)
    bsd := lcg(1103515245, 12345, 1<<31, seed)
    msf := msg(seed)
    fmt.Println("       BSD  Microsoft")
    for i := 0; i < 5; i++ {
        fmt.Printf("%10d    %5d\n", bsd(), msf())
    }
}

func main() {
    example(0)
    example(1)
}
```

Output:

```
With seed = 0
       BSD  Microsoft
     12345       38
1406932606     7719
 654583775    21238
1449466924     2437
 229283573     8855

With seed = 1
       BSD  Microsoft
1103527590       41
 377401575    18467
 662824084     6334
1147902781    26500
2035015474    19169
```

Haskell

```
bsd n = r:bsd r where r = ((n * 1103515245 + 12345) `rem` 2^31)
msr n = (r `div` 2^16):msr r where r = (214013 * n + 2531011) `rem` 2^31

main = do
        print $ take 10 $ bsd 0 -- can take seeds other than 0, of course
        print $ take 10 $ msr 0
```

Icon and Unicon

The following LCRNG's behave in the same way maintaining the state (seed) from round to round. There is an srand procedure for each lcrng that maintains the seed state and allows the user to assign a new state.

```
link printf

procedure main()
   printf("       BSD        MS\n")
   every 1 to 10 do
      printf("%10s %10s\n",rand_BSD(),rand_MS())
end

procedure srand_BSD(x)             #: seed random
static seed
   return seed := \x | \seed | 0   # parm or seed or zero if none
end

procedure rand_BSD()               #: lcrng
   return srand_BSD((1103515245 * srand_BSD() + 12345) % 2147483648)
end

procedure srand_MS(x)              #: seed random
static seed
   return seed := \x | \seed | 0   # parm or seed or zero if none
end

procedure rand_MS()                #: lcrng
```

```
   return ishift(srand_MS((214013 * srand_MS() + 2531011) % 2147483648),-16)
end
```

Library: Icon Programming Library

printf.icn provides printf (http://www.cs.arizona.edu/icon/library/src/procs/printf.icn)

J

Solution:

```
lcg=: adverb define
 0 m lcg y                    NB. default seed of 0
:
 'a c mod'=. x: m
 }. (mod | c + a * ])^:(<y+1) x
)

rand_bsd=: (1103515245 12345 , <.2^31) lcg
rand_ms=: (2^16) <.@:%~ (214013 2531011 , <.2^31) lcg
```

Example Use:

```
   rand_bsd 10
12345 1406932606 654583775 1449466924 229283573 1109335178 1051550459 1293799192 794471793 551188310
   654583775 rand_bsd 4
1449466924 229283573 1109335178 1051550459
   rand_ms 10
38 7719 21238 2437 8855 11797 8365 32285 10450 30612
   1 rand_ms 5                   NB. seed of 1
41 18467 6334 26500 19169
```

K

```
   bsd:{1_ y{((1103515245*x)+12345)!(_2^31)}\x}
   ms:{1_(y{_(((214013*x)+2531011)!(_2^31))}\x)%(_2^16)}

   bsd[0;10]
12345 1406932606 654583775 1449466924 229283573 1109335178 1051550459 1293799192 794471793 551188310
   ms[0;10]
38 7719 21238 2437 8855 11797 8365 32285 10450 30612
```

Liberty BASIC

```
'by default these are 0
global BSDState
global MSState
```

```
for i = 1 to 10
    print randBSD()
next i

print

for i = 1 to 10
    print randMS()
next i

function randBSD()
    randBSD = (1103515245 * BSDState + 12345) mod (2 ^ 31)
    BSDState = randBSD
end function

function randMS()
    MSState = (214013 * MSState + 2531011) mod (2 ^ 31)
    randMS = int(MSState / 2 ^ 16)
end function
```

Logo

Note that, perhaps ironically, UCBLogo, as of version 6.0, doesn't generate the proper output from the BSD constants; it uses double-precision floating point, which is not enough for some of the intermediate products. In UCBLogo, the BSD series deviates starting with the third value (see sample output below).

```
; Configuration parameters for Microsoft and BSD implementations
make "LCG_MS [214013 2531011 65536 2147483648]
make "LCG_BSD [1103515245 12345 1 2147483648]

; Default seed is 0
make "_lcg_value 0

; set the seed
to lcg_seed :seed
  make "_lcg_value :seed
end

; generate the next number in the series using the given parameters
to lcg_rand [:config :LCG_MS]
  local "a local "c local "d local "m
  foreach [a c d m] [
    make ? item # :config
  ]
  make "_lcg_value (modulo (sum (product :a :_lcg_value) :c) :m)
  output int quotient :_lcg_value :d
end

foreach (list :LCG_BSD :LCG_MS) [
  lcg_seed 0
  repeat 10 [
    print (lcg_rand ?)
  ]
  print []
]
bye
```

Output:

```
12345
1406932606
654583775
1449466924
229283573
1109335178
1051550459
1293799192
794471793
551188310

38
7719
21238
2437
8855
11797
8365
32285
10450
30612
```

UCBLogo output for the BSD section:

```
12345
1406932606
654583808
1358247936
2138638336
1459132416
1445521408
370866176
1896597568
1518859008
```

Mathematica

```
BSDrand[x_] := Mod[x*1103515245 + 12345, 2147483648]
NestList[BSDrand, 0, 10]
-> {0, 12345, 1406932606, 654583775, 1449466924, 229283573, 1109335178, 1051550459, 1293799192, 79447179

MSrand[x_] := Mod[x*214013 + 2531011, 2147483648]
BitShiftRight[ NestList[MSrand, 0, 10], 16]
-> {0, 38, 7719, 21238, 2437, 8855, 11797, 8365, 32285, 10450, 30612}
```

Maxima

```
seed: 0$
ms_rand() := quotient(seed: mod(214013 * seed + 2531011, 2147483648), 65536)$
makelist(ms_rand(), 20); /* see http://oeis.org/A096558 */

[38, 7719, 21238, 2437, 8855, 11797, 8365, 32285, 10450, 30612, 5853, 28100, 1142, 281,
```

```
20537, 15921, 8945, 26285, 2997, 14680]

seed: 0$
bsd_rand() := seed: mod(1103515245 * seed + 12345, 2147483648)$
makelist(bsd_rand(), 20); /* see http://www.randomwalk.de/scimath/prngseqs.txt */

[12345, 1406932606, 654583775, 1449466924, 229283573, 1109335178, 1051550459,
1293799192, 794471793, 551188310, 803550167, 1772930244, 370913197, 639546082, 1381971571,
1695770928, 2121308585, 1719212846, 996984527, 1157490780]
```

PARI/GP

Note that up to PARI/GP version 2.3.0, `random()` used a linear congruential generator.

```
BSDseed=Mod(1,1<<31);
MSFTseed=Mod(1,1<<31);
BSD()=BSDseed=1103515245*BSDseed+12345;lift(BSDseed);
MSFT()=MSFTseed=214013*MSFTseed+2531011;lift(MSFTseed)%(1<<31);
```

Pascal

```
Program LinearCongruentialGenerator(output);

var
  x1, x2: int64;

function bsdrand: longint;
  const
    a = 1103515245;
    c = 12345;
    m = 2147483648;
  begin
    x1 := (a * x1 + c) mod m;
    bsdrand := x1;
  end;

function msrand: longint;
  const
    a = 214013;
    c = 2531011;
    m = 2147483648;
  begin
    x2 := (a * x2 + c) mod m;
    msrand := x2 div 65536;
  end;

var
  i: longint;
begin
  writeln('      BSD           MS');
  x1 := 0;
  x2 := 0;
  for i := 1 to 10 do
    writeln(bsdrand:12, msrand:12);
end.
```

Output:

```
     BSD           MS
      12345        7584
 1124652145        3277
 1499545833        3067
 1558406049       31446
  696007321       13069
   56579025       17343
 1312705865        2510
  811881729        5264
 1301653753       21298
 1318262577       27689
```

Perl

Creates a magic scalar whose value is next in the LCG sequence when read.

```
use strict;
package LCG;

use overload '0+'  => \&get;

use integer;
sub gen_bsd { (1103515245 * shift() + 12345) % (1 << 31) }

sub gen_ms  {
        my $s = (214013 * shift() + 2531011) % (1 << 31);
        $s, $s / (1 << 16)
}

sub set { $_[0]->{seed} = $_[1] } # srand
sub get {
        my $o = shift;
        ($o->{seed}, my $r) = $o->{meth}->($o->{seed});
        $r //= $o->{seed}
}

sub new {
        my $cls = shift;
        my %opts = @_;
        bless {
                seed => $opts{seed},
                meth => $opts{meth} eq 'MS' ? \&gen_ms : \&gen_bsd,
        }, ref $cls || $cls;
}

package main;

my $rand = LCG->new;

print "BSD:\n";
print "$rand\n" for 1 .. 10;

$rand = LCG->new(meth => 'MS');

print "\nMS:\n";
print "$rand\n" for 1 .. 10;
```

output

```
BSD:
12345
1406932606
654583775
1449466924
229283573
1109335178
1051550459
1293799192
794471793
551188310

MS:
38
7719
21238
2437
8855
11797
8365
32285
10450
30612
```

Perl 6

Define subroutines implementing the LCG algorithm for each version then use those to generate lazy infinite lists of values and return the first 10 values from each.

```
my $mod = 2**31;
sub bsd ($seed) { ( 1103515245 * $seed + 12345   ) % $mod };
sub ms  ($seed) { ( 214013     * $seed + 2531011 ) % $mod };

say 'BSD LCG first 10 values:';
.say for ( 0.&bsd, -> $seed { $seed.&bsd } ... * )[^10];

say "\nMS LCG first 10 values:";
($_ +> 16).say for ( 0.&ms, -> $seed { $seed.&ms } ... * )[^10];
```

```
BSD LCG first 10 values:
12345
1406932606
654583775
1449466924
229283573
1109335178
1051550459
1293799192
794471793
551188310

MS LCG first 10 values:
38
7719
```

```
21238
2437
8855
11797
8365
32285
10450
30612
```

PHP

Works with: PHP version 5.3+

```
<?php
function bsd_rand($seed) {
    return function() use (&$seed) {
        return $seed = (1103515245 * $seed + 12345) % (1 << 31);
    };
}

function msvcrt_rand($seed) {
    return function() use (&$seed) {
        return ($seed = (214013 * $seed + 2531011) % (1 << 31)) >> 16;
    };
}

$lcg = bsd_rand(0);
echo "BSD ";
for ($i = 0; $i < 10; $i++)
    echo $lcg(), " ";
echo "\n";

$lcg = msvcrt_rand(0);
echo "Microsoft ";
for ($i = 0; $i < 10; $i++)
    echo $lcg(), " ";
echo "\n";
?>
```

PicoLisp

```
(zero *BsdSeed *MsSeed)

(de bsdRand ()
   (setq *BsdSeed
      (& (+ 12345 (* 1103515245 *BsdSeed)) `(dec (** 2 31))) ) )

(de msRand ()
   (>> 16
      (setq *MsSeed
         (& (+ 2531011 (* 214013 *MsSeed)) `(dec (** 2 31))) ) ) )
```

Output:

```
: (do 7 (printsp (bsdRand)))
```

```
12345 1406932606 654583775 1449466924 229283573 1109335178 1051550459 -> 1051550459

: (do 12 (printsp (msRand)))
38 7719 21238 2437 8855 11797 8365 32285 10450 30612 5853 28100 -> 28100
```

PL/I

```
(nofixedoverflow, nosize):
LCG: procedure options (main);
   declare i fixed binary;

   put skip list ('BSD', 'MS');
   do i = 1 to 20;
      put skip list (BSD(), MS());
   end;

bsd: procedure returns (fixed binary (31));
    declare const fixed binary static initial (12345);
    declare s fixed binary (31) static initial (123456789);

    s = s * 1103515245 + const;
    s = isrl(isll(s,1), 1);
    return (s);
end bsd;
ms: procedure returns (fixed binary (15));
    declare const fixed binary (31) static initial (2531011);
    declare s     fixed binary (31) static initial (123456789);

    s = s * 214013 + const;
    s = isrl(isll(s,1), 1);
    return (isrl(s,16));

end ms;

end LCG;
```

OUTPUT:

```
BSD                   MS
     231794730             13259
    1126946331             26974
    1757975480             13551
     850994577             30354
    1634557174             18709
     707246327             15861
    1397699428             16906
    1035569613             21981
    1904890498              8603
    1335160211             12911
    1434329552             18110
    1273099721              3228
    1250890958             27918
    1016516591             17989
    1097566972             22768
     436938117             23599
    1175171034              7712
```

```
   1059748875              15601
    308566760               7038
    534615297              21512
```

PureBasic

```
Procedure ms_LCG(seed.q = -1)
  Static state.q
  If seed >= 0
    state = seed
  Else
    state = (state * 214013 + 2531011) % (1 << 31)
    ProcedureReturn state >> 16
  EndIf
EndProcedure

Procedure.q bsd_LCG(seed.q = -1)
  Static state.q
  If seed >= 0
    state = seed
  Else
    state = (state * 1103515245 + 12345) % (1 << 31)
    ProcedureReturn state
  EndIf
EndProcedure

If OpenConsole()
  Define i
  PrintN("BSD (seed = 1)")
  bsd_LCG(1)
  For i = 1 To 5
    PrintN(Str(bsd_LCG()))
  Next

  PrintN(#CRLF$ + "MS (seed = 1)")
  ms_LCG(1)
  For i = 1 To 5
    PrintN(Str(ms_LCG()))
  Next

  Print(#CRLF$ + #CRLF$ + "Press ENTER to exit"): Input()
  CloseConsole()
EndIf
```

Sample output:

```
BSD (seed = 1)
1103527590
377401575
662824084
1147902781
2035015474

MS (seed = 1)
41
18467
6334
26500
```

```
19169
```

Python

```
def bsd_rand(seed):
   def rand():
      rand.seed = (1103515245*rand.seed + 12345) & 0x7fffffff
      return rand.seed
   rand.seed = seed
   return rand

def msvcrt_rand(seed):
   def rand():
      rand.seed = (214013*rand.seed + 2531011) & 0x7fffffff
      return rand.seed >> 16
   rand.seed = seed
   return rand
```

Works with: Python version 3.x

```
def bsd_rand(seed):
   def rand():
      nonlocal seed
      seed = (1103515245*seed + 12345) & 0x7fffffff
      return seed
   return rand

def msvcrt_rand(seed):
   def rand():
      nonlocal seed
      seed = (214013*seed + 2531011) & 0x7fffffff
      return seed >> 16
   return rand
```

Racket

The following solution uses generators and transcribes the mathematical formulas above directly. It does not attempt to be efficient.

```

#lang racket
(require racket/generator)

(define (bsd-update state_n)
  (modulo (+ (* 1103515245 state_n) 12345)
          (expt 2 31)))

(define (ms-update state_n)
  (modulo (+ (* 214013 state_n) 2531011)
          (expt 2 31)))

(define ((rand update ->rand) seed)
  (generator ()
   (let loop ([state_n seed])
```

```
    (define state_n+1 (update state_n))
    (yield (->rand state_n+1))
    (loop state_n+1))))

(define bsd-rand (rand bsd-update identity))
(define ms-rand (rand ms-update (λ (x) (quotient x (expt 2 16)))))
```

REXX

```
/*REXX program congruential generator which simulates the old BSD &  MS */
/*  random number generators.   BSD= 0──►(2**31)-1,   MS= 0──►(2**16)-1 */
numeric digits 20                      /*enough digits for the multiply.*/

  do seed=0 to 1;   bsd=seed;    ms=seed;   say center('seed='seed,79,'═')
        do j=1 for 20;      jjj=right(j,3)
        bsd =    (1103515245 * bsd   +     12345)   //  2**31
        ms  =    (    214013 *  ms   +   2531011)   //  2**31
        say  'state'   jjj    " BSD"   right(bsd,     11)    left('',18),
                              "MS"   right( ms,     11)     left('',5),
                             "rand"   right(ms%2**16, 6)
        end   /*j*/
  end         /*seed*/
                                       /*stick a fork in it, we're done.*/
```

output

```
═══════════════════════════════════seed=0═══════════════════════════════════
state   1  BSD       12345                    MS     2531011       rand     38
state   2  BSD  1406932606                    MS   505908858       rand   7719
state   3  BSD   654583775                    MS  1391876949       rand  21238
state   4  BSD  1449466924                    MS   159719620       rand   2437
state   5  BSD   229283573                    MS   580340855       rand   8855
state   6  BSD  1109335178                    MS   773150046       rand  11797
state   7  BSD  1051550459                    MS   548247209       rand   8365
state   8  BSD  1293799192                    MS  2115878600       rand  32285
state   9  BSD   794471793                    MS   684884587       rand  10450
state  10  BSD   551188310                    MS  2006221698       rand  30612
state  11  BSD   803550167                    MS   383622205       rand   5853
state  12  BSD  1772930244                    MS  1841626636       rand  28100
state  13  BSD   370913197                    MS    74896543       rand   1142
state  14  BSD   639546082                    MS    18439398       rand    281
state  15  BSD  1381971571                    MS  1345953809       rand  20537
state  16  BSD  1695770928                    MS  1043415696       rand  15921
state  17  BSD  2121308585                    MS   586225427       rand   8945
state  18  BSD  1719212846                    MS  1722639754       rand  26285
state  19  BSD   996984527                    MS   196417061       rand   2997
state  20  BSD  1157490780                    MS   962080852       rand  14680
═══════════════════════════════════seed=1═══════════════════════════════════
state   1  BSD  1103527590                    MS     2745024       rand     41
state   2  BSD   377401575                    MS  1210316419       rand  18467
state   3  BSD   662824084                    MS   415139642       rand   6334
state   4  BSD  1147902781                    MS  1736732949       rand  26500
state   5  BSD  2035015474                    MS  1256316804       rand  19169
state   6  BSD   368800899                    MS  1030492215       rand  15724
state   7  BSD  1508029952                    MS   752224798       rand  11478
state   8  BSD   486256185                    MS  1924036713       rand  29358
state   9  BSD  1062517886                    MS  1766988168       rand  26962
```

```
state  10  BSD   267834847                  MS  1603301931       rand  24464
```

Ruby

You can create multiple instances of LCG::Berkeley or LCG::Microsoft. Each instance privately keeps the original seed in @seed, and the current state in @r. Each class resembles the core Random class, but with fewer features. The .new method takes a seed. The #rand method returns the next random number. The #seed method returns the original seed.

```ruby
module LCG
  module Common
    # The original seed of this generator.
    attr_reader :seed

    # Creates a linear congruential generator with the given _seed_.
    def initialize(seed)
      @seed = @r = seed
    end
  end

  # LCG::Berkeley generates 31-bit integers using the same formula
  # as BSD rand().
  class Berkeley
    include Common
    def rand
      @r = (1103515245 * @r + 12345) & 0x7fff_ffff
    end
  end

  # LCG::Microsoft generates 15-bit integers using the same formula
  # as rand() from the Microsoft C Runtime.
  class Microsoft
    include Common
    def rand
      @r = (214013 * @r + 2531011) & 0x7fff_ffff
      @r >> 16
    end
  end
end
```

The next example sets the seed to 1, and prints the first 5 random numbers.

```ruby
lcg = LCG::Berkeley.new(1)
p (1..5).map {lcg.rand}
# prints [1103527590, 377401575, 662824084, 1147902781, 2035015474]

lcg = LCG::Microsoft.new(1)
p (1..5).map {lcg.rand}
# prints [41, 18467, 6334, 26500, 19169]
```

Scala

```
object LinearCongruentialGenerator {
  def bsdRandom(rseed:Int):Iterator[Int]=new Iterator[Int]{
    var seed=rseed
    override def hasNext:Boolean=true
    override def next:Int={seed=(seed * 1103515245 + 12345) & Int.MaxValue; seed}
  }

  def msRandom(rseed:Int):Iterator[Int]=new Iterator[Int]{
    var seed=rseed
    override def hasNext:Boolean=true
    override def next:Int={seed=(seed * 214013 + 2531011) & Int.MaxValue; seed >> 16}
  }

  def toString(it:Iterator[Int], n:Int=20)=it take n mkString ", "

  def main(args:Array[String]){
    println("-- seed 0 --")
    println("BSD: "+ toString(bsdRandom(0)))
    println("MS : "+ toString(msRandom(0)))

    println("-- seed 1 --")
    println("BSD: "+ toString(bsdRandom(1)))
    println("MS : "+ toString( msRandom(1)))
  }
}
```

Output:

```
-- seed 0 --
BSD: 12345, 1406932606, 654583775, 1449466924, 229283573, 1109335178, 1051550459, 1293799192,
794471793, 551188310, 803550167, 1772930244, 370913197, 639546082, 1381971571, 1695770928,
2121308585, 1719212846, 996984527, 1157490780

MS : 38, 7719, 21238, 2437, 8855, 11797, 8365, 32285, 10450, 30612, 5853, 28100, 1142, 281, 20537,
15921, 8945, 26285, 2997, 14680

-- seed 1 --
BSD: 1103527590, 377401575, 662824084, 1147902781, 2035015474, 368800899, 1508029952, 486256185,
1062517886, 267834847, 180171308, 836760821, 595337866, 790425851, 2111915288, 1149758321,
1644289366, 1388290519, 1647418052, 1675546029

MS : 41, 18467, 6334, 26500, 19169, 15724, 11478, 29358, 26962, 24464, 5705, 28145, 23281, 16827,
9961, 491, 2995, 11942, 4827, 5436
```

Scheme

```
(define ((bsd-rand seed)) (set! seed (remainder (+ (* 1103515245 seed) 12345) 2147483648)) seed)

(define ((msvcrt-rand seed)) (set! seed (remainder (+ (* 214013 seed) 2531011) 2147483648)) (quotient se

; auxiliary function to get a list of 'n random numbers from generator 'r
(define (rand-list r n) = (if (zero? n) '() (cons (r) (rand-list r (- n 1)))))

(rand-list (bsd-rand 0) 10)
; (12345 1406932606 654583775 1449466924 229283573 1109335178 1051550459 1293799192 794471793 551188310)

(rand-list (msvcrt-rand 0) 10)
```

```
; (38 7719 21238 2437 8855 11797 8365 32285 10450 30612)
```

Seed7

Seed7 provides also a random number generator. The random function is overloaded for many types. E.g.: The library integer.s7i (http://seed7.sourceforge.net/libraries/integer.htm) defines rand(lower, upper) (http://seed7.sourceforge.net/libraries/integer.htm#rand%28in_integer,in_integer%29) . The parameters specifiy the lower and upper bound of the desired random value. The library array.s7i (http://seed7.sourceforge.net/libraries/array.htm) defines rand(arr) (http://seed7.sourceforge.net/libraries/array.htm#rand%28in_arrayType%29) . This function selects a random element from an array.

```
$ include "seed7_05.s7i";
  include "bigint.s7i";

var bigInteger: bsdSeed is 0_;
var bigInteger: msSeed is 0_;

const func integer: bsdRand is func
  result
    var integer: bsdRand is 0;
  begin
    bsdSeed := (1103515245_ * bsdSeed + 12345_) mod 2147483648_;
    bsdRand := ord(bsdSeed);
  end func;

const func integer: msRand is func
  result
    var integer: msRand is 0;
  begin
    msSeed := (214013_ * msSeed + 2531011_) mod 2147483648_;
    msRand := ord(msSeed) mdiv 65536;
  end func;

const proc: main is func
  local
    var integer: i is 0;
  begin
    writeln("         BSD          MS");
    for i range 1 to 10 do
      writeln(bsdRand lpad 12 <& msRand lpad 12);
    end for;
  end func;
```

Output:

```
         BSD          MS
       12345          38
  1406932606        7719
   654583775       21238
  1449466924        2437
   229283573        8855
```

```
 1109335178       11797
 1051550459        8365
 1293799192       32285
  794471793       10450
  551188310       30612
```

Tcl

Using an object-oriented solution, inspired by (but not a translation of) the Ruby solution above.

```
package require Tcl 8.6

# General form of a linear-congruential RNG
oo::class create LCRNG {
    variable seed A B C D
    constructor {init a b c d} {
        if {$init < 1} {set init [clock clicks]}
        variable seed $init A $a B $b C $c D $d
    }
    method rand {} {
        set seed [expr {($A * $seed + $B) % $C}]
        return [expr {$seed / $D}]
    }
    method srand x {
        set seed $x
    }
}
# Subclass to introduce constants
oo::class create BSDRNG {
    superclass LCRNG
    constructor {{initialSeed -1}} {
        next $initialSeed 1103515245 12345 [expr {2**31}] 1
    }
}
oo::class create MSRNG {
    superclass LCRNG
    constructor {{initialSeed -1}} {
        next $initialSeed 214013 2531011 [expr {2**31}] [expr {2**16}]
    }
}
```

Demo code:

```
proc sample rng {foreach - {1 2 3 4 5} {lappend r [$rng rand]}; join $r ", "}
puts BSD:\t\[[sample [BSDRNG new 1]]\]
puts MS:\t\[[sample [MSRNG new 1]]\]
```

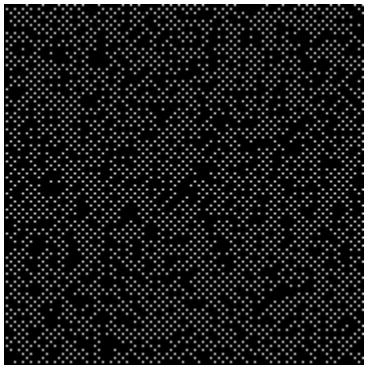
Output:

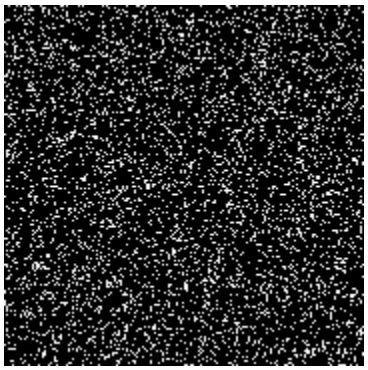
```
BSD:    [1103527590, 377401575, 662824084, 1147902781, 2035015474]
MS:     [41, 18467, 6334, 26500, 19169]
```

XPL0

It's not easy just by looking at the numbers generated if they are sufficiently random. You might notice that the BSD numbers alternate odd and even, which is pretty bad. A simple but effective test is to simulate falling snowflakes.

```
include c:\cxpl\codes;
int R;

func BSD;
[R:= (1103515245*R + 12345) & $7FFF_FFFF;
return R;
]; \BSD


func MSFT;
[R:= (214013*R + 2531011) & $7FFF_FFFF;
return R>>16;
]; \MSFT


int N;
[SetVid(4);             \320x200x2 graphics
R:= 0;                  \initialize seed
for N:= 0 to 5000 do
        Point(rem(BSD/180), rem(BSD/180), 3);
N:= ChIn(1);            \wait for keystoke

SetVid(4);              \320x200x2 graphics
R:= 0;                  \initialize seed
for N:= 0 to 5000 do
        Point(rem(MSFT/180), rem(MSFT/180), 3);
N:= ChIn(1);            \wait for keystoke
SetVid(3);              \restore normal text mode
]
```





Retrieved from "http://rosettacode.org/mw/index.php?title=Linear_congruential_generator&oldid=181731"
Categories: Programming Tasks | Randomness | Ada | AutoHotkey | BBC BASIC | Bc | Bracmat | C | C++ | Clojure | Common Lisp | C sharp | D | Dc | F Sharp | Forth | Fortran | Go | Haskell | Icon | Unicon | Icon Programming Library | J | K | Liberty BASIC | Logo | Mathematica | Maxima | PARI/GP | Pascal | Perl | Perl 6 | PHP | PicoLisp | PL/I | PureBasic | Python | Racket | REXX | Ruby | Scala | Scheme | Seed7 | Tcl | XPL0
