generate random double numbers in c++

How to generate random numbers between two doubles in c++ , these numbers should look like xxxxx,yyyyy .

c++ random

edited Dec 18 '16 at 11:13

asked Apr 24 '10 at 12:56

Radi
2,314 14 46 77

5 "these number should look like xxxxx,yyyyy". How to generate random doubles, and how to format doubles as strings, are completely separate issues. – Steve Jessop Apr 24 '10 at 13:09

And come to think of it alternatively: generating evenly-distributed *doubles* and generating evenly-distributed *decimals* are somewhat different, although related, tasks. – Steve Jessop Apr 24 '10 at 13:24

Generating evenly-distributed integers is more closely related to the decimals problem. – Potatoswatter Apr 24 '10 at 19:56

7 Answers

Here's how

```
double fRand(double fMin, double fMax)
{
    double f = (double)rand() / RAND_MAX;
    return fMin + f * (fMax - fMin);
}
```

Remember to call srand() with a proper seed each time your program starts.

edited Apr 24 '10 at 13:39

Frank
24k 68 186 282

answered Apr 24 '10 at 13:10

rep_movsd
4,131 1 21 27

2 to exclude max (often done): add +1 to RAND_MAX – KillianDS Apr 24 '10 at 13:12

7 If you add 1 to RAND_MAX, do so carefully, since it might be equal to INT_MAX. `double f = rand() / (RAND_MAX + 1.0);` – Steve Jessop Apr 24 '10 at 13:22

7 Note that the randomness of this can be limited. The range xxxxx,yyyyy suggests 10 decimal digits. There are plenty of systems where RAND_MAX is smaller than 10^10. This would mean that some numbers in that range have `p(xxxxx,yyyyy)==0.0` – MSalters Apr 26 '10 at 12:46

2 You should avoid rand() if possible. See the other answer for a C++11 or TR1 solution. – jfritz42 Aug 4 '14 at 23:01

1 @ChamilaWijayarathna, You need to include cstdlib – Veridian Mar 30 '15 at 3:31

This solution requires C++11 (or TR1).

```
#include <random>

int main()
{
   double lower_bound = 0;
   double upper_bound = 10000;
   std::uniform_real_distribution<double> unif(lower_bound,upper_bound);
   std::default_random_engine re;
   double a_random_double = unif(re);

   return 0;
}
```

For more details see John D. Cook's "Random number generation using C++ TR1".

See also Stroustrup's "Random number generation".

edited Nov 16 '13 at 9:47

answered Feb 17 '12 at 8:04

Alessandro Jacopson
8,233 6 55 98

2 You might want to update this with a more recent cppreference document, which is pretty good. – Shafik Yaghmour Jul 23 '13 at 3:07

If accuracy is an issue here you can create random numbers with a finer graduation by randomizing the significant bits. Let's assume we want to have a double between 0.0 and 1000.0.

On MSVC (12 / Win32) RAND_MAX is 32767 for example.

If you use the common `rand()/RAND_MAX` scheme your gaps will be as large as

```
1.0 / 32767.0 * ( 1000.0 - 0.0) = 0.0305 ...
```

In case of IEE 754 double variables (53 significant bits) and 53 bit randomization the smallest possible randomization gap for the 0 to 1000 problem will be

```
2^-53 * (1000.0 - 0.0) = 1.110e-13
```

and therefore significantly lower.

The downside is that 4 rand() calls will be needed to obtain the randomized integral number (assuming a 15 bit RNG).

```
double random_range (double const range_min, double const range_max)
{
  static unsigned long long const mant_mask53(9007199254740991);
  static double const i_to_d53(1.0/9007199254740992.0);
  unsigned long long const r( (unsigned long long(rand()) | (unsigned long
long(rand()) << 15) | (unsigned long long(rand()) << 30) | (unsigned long
long(rand()) << 45)) & mant_mask53 );
  return range_min + i_to_d53*double(r)*(range_max-range_min);
}
```

If the number of bits for the mantissa or the RNG is unknown the respective values need to be obtained within the function.

```
#include <limits>
using namespace std;
double random_range_p (double const range_min, double const range_max)
{
  static unsigned long long const num_mant_bits(numeric_limits<double>::digits),
ll_one(1),
    mant_limit(ll_one << num_mant_bits);
  static double const i_to_d(1.0/double(mant_limit));
  static size_t num_rand_calls, rng_bits;
  if (num_rand_calls == 0 || rng_bits == 0)
  {
    size_t const rand_max(RAND_MAX), one(1);
    while (rand_max > (one << rng_bits))
    {
      ++rng_bits;
    }
    num_rand_calls = size_t(ceil(double(num_mant_bits)/double(rng_bits)));
  }
  unsigned long long r(0);
  for (size_t i=0; i<num_rand_calls; ++i)
  {
    r |= (unsigned long long(rand()) << (i*rng_bits));
  }
  r = r & (mant_limit-ll_one);
  return range_min + i_to_d*double(r)*(range_max-range_min);
}
```

Note: I don't know whether the number of bits for unsigned long long (64 bit) is greater than the number of double mantissa bits (53 bit for IEE 754) on all platforms or not. It would probably be "smart" to include a check like `if (sizeof(unsigned long long)*8 > num_mant_bits) ...` if this is not the case.

answered Apr 8 '13 at 0:37

Pixelchemist
12.2k 4 22 53

This snippet is straight from Stroustrup's *The C++ Programming Language (4th Edition)*, §40.7;
it requires C++11:

```
#include <functional>
#include <random>

class Rand_double
{
public:
    Rand_double(double low, double high)
    :r(std::bind(std::uniform_real_distribution<>
(low,high),std::default_random_engine())){}

    double operator()(){ return r(); }

private:
    std::function<double()> r;
};

#include <iostream>
int main() {
    // create the random number generator:
    Rand_double rd{0,0.5};

    // print 10 random number between 0 and 0.5
    for (int i=0;i<10;++i){
        std::cout << rd() << ' ';
    }
    return 0;
}
```

answered Nov 16 '13 at 9:30

Alessandro Jacopson
8,233 6 55 98

This should be performant, thread-safe and flexible enough for many uses:

```
#include <random>
#include <iostream>

template<typename Numeric, typename Generator = std::mt19937>
Numeric random(Numeric from, Numeric to)
{
    thread_local static Generator gen(std::random_device{}());

    using dist_type = typename std::conditional
    <
        std::is_integral<Numeric>::value
        , std::uniform_int_distribution<Numeric>
        , std::uniform_real_distribution<Numeric>
    >::type;

    thread_local static dist_type dist;

    return dist(gen, typename dist_type::param_type{from, to});
}

int main(int, char*[])
{
    for(auto i = 0U; i < 20; ++i)
        std::cout << random<double>(0.0, 0.3) << '\n';
}
```

answered Feb 28 '16 at 20:04

Galik
22.8k 3 29 65

---

something like this:

```
#include <iostream>
#include <time.h>

using namespace std;

int main()
{
    const long max_rand = 1000000L;
    double x1 = 12.33, x2 = 34.123, x;

    srandom(time(NULL));

    x = x1 + ( x2 - x1) * (random() % max_rand) / max_rand;

    cout << x1 << " <= " << x << " <= " << x2 << endl;

    return 0;
}
```

answered Apr 24 '10 at 13:13

Oleg Razgulyaev
2,772 2 17 27

1 "(random() % max_rand)" = "random()" (i.e. 3 % 7 = 3). This would be a wasted processing step. – Zak Jan 25 '12 at 21:53

- This is for c++

```
#include "stdafx.h"
#include "iostream"
#include "ctime"

using namespace std;

double getRandom(double min, double max)
{
    double before = rand() % (int)max + (int)min;
    double after = (double)rand() / RAND_MAX;
    double result = before + after;
    if (result < min || result > max) {
        result = getRandom(min, max);
    }
    return result;
}

int main()
{
    srand (time(NULL));
    for (int i = 0; i < 100; i++) {
        double number = getRandom(-1.57079632679, 1.57079632679);
        cout << number << endl;
    }
    system("pause");
}
```

answered Mar 1 at 15:13

Marco Salerno
799 1 11