Generating Pseudo-Random Numbers: Integers and Floats

Dr. Kevin Farrell

September 2024

Table of Contents

1 http://creativecommons.org/
2 http://creativecommons.org/licenses/by-nc-nd/3.0/

1. Random numbers are “Pseudo”

Random numbers are simply a very long sequence of numbers stored somewhere on your computer; for eg: here’s a short sequence:

98, 50, 60, 35, 6, 13, 17, 44, 72, ...

We can ‘generate’ four random numbers by picking the first four numbers here:

98, 50, 60, 35

We can ‘generate’ four other random numbers by picking the numbers #2 to #5 here:

50, 60, 35, 6

We can see that the sequence of numbers isn’t very random since there is some common numbers between the first set of four numbers ‘generated’ and the second set of four numbers ‘generated’. This is how random numbers are generated on a computer; hence, we refer to them as pseudo-random numbers, because they are not really random at all!

The rand() function generates a pseudo-random **integer** number between 0 and RAND_MAX, where RAND_MAX is a number defined in the OS. The function rand() picks a number from the very long sequence stored on the computer.

There is a **defined** starting-point to read from the long sequence. If we write a program which calls rand() 10 times to generate 10 random numbers for example, rand() will start at that defined point in the sequence, and list the 10 numbers in the sequence from that point. If we run our program several times, each time it will give us **exactly** the same 10 ‘random’ numbers.

To make our random numbers “more-random”, we need to start reading numbers from the sequence at a **different** starting point each time we run your program.

To do this, use the **srand()** function. This function takes one argument: an unsigned integer we which we refer to as the “seed” for the sequence. If the seed is the same each time, then the starting point in the sequence that rand() will use, will be exactly the same. So, we have to make sure the seed is different each time. Often, we simply use the time of day, cast as an unsigned int, as the seed:

```
(unsigned int)time(NULL)
```

So, in C, our call to srand() will look like this:

```
srand((unsigned int)time(NULL));
```

which generates a seed – a starting point in the sequence – which is different each time you run your program, because the time() function, which gives the time in seconds since 1 January 1970, gives us a different value each time it is run.

2. How to generate pseudo-random integers

To generate a random **integer** between 0 and 99, use:

```
    srand((unsigned int)time(NULL));

    int myrandom_int = rand()%100;
```

rand() returns a number between 0 and RAND_MAX. **RAND_MAX** is a huge number, defined in the `stdlib.h` header file that comes with the C compiler. The definition is as follows:

```
    #define RAND_MAX 2147483647
```

So, how is `rand()%100` generating a pseudo-random integer between 0 and 99? To aid our understanding, let's look at the **modulo** function, represented by the **%** symbol in C. That is:

x % y means **"x modulo y"**

Rule: x % y is the Remainder we obtain after dividing x by y.

Example 1: 6500 modulo 100 = 6500 % 100 = 0, because 6500 / 100 = 65, Remainder = 0

Example 2: What is: 7899 % 100 ?

Well, 7899/100 = 78, Remainder = 99 => 7899%100 = 99

Example 3: What is 102 % 100 ?

Well, 102/100 = 1, Remainder = 2 => 102%100 = 2

Reminder: Rule: x % y is the Remainder we obtain after dividing x by y.

To generate a pseudo-random integer between 1 and 100 instead of between 0 and 99, simply add 1 in our code at the start of this section:

```
    srand((unsigned int)time(NULL));

    int myrandom_int = rand()%100 + 1;
```

3. How to generate pseudo-random floating-point numbers

3.1. Properties of rand() and RAND_MAX

Question: What range of values will the following take on: **rand() / RAND_MAX** (i.e. in this ***division***)?

Lowest value = **0 / RAND_MAX = 0**

Highest value = **RAND_MAX / RAND_MAX = 1**

Answer: range of values is between 0 and 1 inclusive.

But rand() returns an int **not** a float

and RAND_MAX is also an int **not** a float

In C, rounding is always downwards – essentially, in C, for an int, any decimal portion of the number is thrown away.

For example:

```
int x = 1/5; // since 1/5 = 0.2, C throws away the ".2" => result: 0

int y = 7899 / RAND_MAX; // a fraction < 1 => result: y = 0

int z = RAND_MAX / RAND_MAX; // exactly 1 => result: z = 1
```

So, to generate random floating point number between 0.0 and 1.0 (inclusive), we **cannot** use:

```
float ranfloat = rand() / RAND_MAX;  //THIS IS WRONG!!!
```

because this is almost always a fraction less than 1, since the rand() function returns an

int < RAND_MAX the vast majority of the time.

The only time rand() / RAND_MAX = 1 is when rand() returns exactly the value RAND_MAX, which is a very rare event.

For example: For simplicity to aid explanation, let's say RAND_MAX = 10,000. Then:

7,899 / RAND_MAX = 7,899/10,000 = 0.7899

But in C, this would equal 0, since C always rounds down fractions; i.e. throws away the portion after the decimal point.

3.2. Generating pseudo-random floating-point numbers between 0.0 and 1.0

Because of the issues with integers and C just discussed in the previous section, we need to use a *different* methodology if we want to generate pseudo-random ***floating-point*** numbers between 0.0 and 1.0:

```
srand((unsigned int)time(NULL));

float ranfloat = (float)rand() / (float)(RAND_MAX);
```

(float)rand() ***casts*** the (integer) output of rand() as floating-point number

(float)(RAND_MAX) ***casts*** the (integer) RAND_MAX as a floating-point number

Now the division is no longer rounded to zero in C because the division is between two floating-point numbers (not between two integers).

So, if rand() returns 7,899 => (float)rand() = 7899.0

If RAND_MAX = 10000 => (float)RAND_MAX = 10000.0

=> (float)rand() / (float)RAND_MAX = 7899.0 / 10000.0 = 0.7899, which is between 0.0 and 1.0.

3.3. Generating pseudo-random floating-point numbers between two different values (other than 0.0 and 1.0)

Hopefully, now you are asking the question: How do we generate pseudo-random floating-point numbers between a different range, other than 0.0 and 1.0?

3.3.1. How do we generate pseudo-random floating-point numbers between 1.0 and 10.0?

Note: In what follows, we have left out the line with **srand()** to simplify things, but the call to **srand()** is still needed when we're actually writing our code.

Recall that the following formula:

```
float ranfloat = ( (float)rand() / (float)(RAND_MAX) )*10.0;
```

gives us numbers from 0.0 to 10.0.

Question: What happens if we add 1 to this?

Answer: It gives us:

numbers from 0.0 + 1.0 to 10.0 + 1.0

= numbers from 1.0 to 11.0

But we want numbers between 1.0 and 10.0. So, how to do it?

When we add a number, we want the upper bound to sum up to 10.0. But, we're adding 1.0 to everything! What if we generated random numbers between 0.0 and some different value? Would that help? Let's look: First, let's generate pseudo-random floating-point numbers between 0.0 and 9.0:

```
    float ranfloat = ( (float)rand() / (float)(RAND_MAX) )*9.0;
```

Now add 1.0 to the Right-Hand side => we get:

numbers between 0.0 + 1.0 and 9.0 + 1.0

= numbers between 1.0 to 10.0

The correct formula is for random floating-point numbers between 1.0 and 10.0 is therefore:

```
    float ranfloat = ( ( (float)rand() / (float)(RAND_MAX) )*9.0 ) + 1.0;
```

3.3.2. How to generate pseudo-random floating point numbers between values 'a' and 'b'?

We give the answer to this question first, and then show that it is true:

```
    float ranfloat = ( ((float)rand() / (float)(RAND_MAX))*(b – a) ) + a;
```

Example #1: Let a = 1.0, b = 10.0. Then to generate numbers between 1.0 and 10.0, we use the formula:

```
    float ranfloat = ( ((float)rand() / (float)(RAND_MAX))*(b – a) ) + a;
```

which becomes:

```
    float ranfloat = (((float)rand() / (float)(RAND_MAX))*(10.0 – 1.0)) + 1.0;
```

which is the same as:

```
    float ranfloat = (((float)rand() / (float)(RAND_MAX))*9.0) + 1.0;
```

which we saw previously is the correct formula to generate pseudo-random floating-point numbers between 1.0 and 10.0.

Example #2: Let a = 3.0, b = 14.0; i.e. generate random floating point numbers between 3.0 and 14.0. We start with our formula:

```
float ranfloat = ( ( (float)rand() / (float)(RAND_MAX) )*(b – a) ) + a;
```

and substitute a = 3.0 and b = 14.0 into it, to get:

```
float ranfloat = (((float)rand() / (float)(RAND_MAX))*(14.0 – 3.0)) + 3.0;
```

which becomes:

```
float ranfloat = ( ( (float)rand() / (float)(RAND_MAX) )*11.0 ) + 3.0;
```

This formula:

```
float ranfloat = ( ( (float)rand() / (float)(RAND_MAX) )*11.0 )
```

gives random floats between: 0.0 and 11.0. If I then add 3.0, I get:

- lowest value of float generated = 0.0 + 3.0 = 3.0
- largest value of float generated = 11.0 + 3.0 = 14.0

which is correct!!!

So, the General Formula to generate floating-point numbers between ‘a’ and ‘b’ (inclusive) is:

```
srand((unsigned int)time(NULL));

float ranfloat = ( ( (float)rand() / (float)(RAND_MAX) )*(b – a) ) + a;
```