Random Number Generation in C and C++

Using rand()

It is often useful to generate random numbers to produce simulations, games or homework problems. One way to generate these numbers in C++ is to use the function **`rand()`.** `Rand` is defined as:

```
#include <cstdlib>
int rand();
```

The **`rand`** function takes no arguments and returns an integer that is a pseudo-random number between 0 and **`RAND_MAX`.** On transformer, **`RAND_MAX`** is **`32767`**. What is a pseudo-random number? It is a number that is not truly random, but appears random. That is, every number between 0 and **`RAND_MAX`** has an equal chance (or probability) of being chosen each time **`rand()`** is called.

For example, the following program might print out:

```
cout <<rand() << endl;
cout <<rand() << endl;
cout <<rand() << endl;

9383
30886
12777
```

Here we "randomly" were given numbers between 0 and `RAND_MAX.` What if we only wanted numbers between 1 and 10? We will need to scale the results. To do this we can use the modulus (%) operator.

Any integer modulus 10 will produce a number from 0-9. In other words, if we divide a number by 10, the remainder has to be from 0 to 9. It's impossible to divide a number by 10 and end up with a remainder bigger than or equal to ten. In our case:

```
9383 % 10 = 3
30886 % 10 = 6
12777 % 10 = 7
```

Consequently, we can take **`rand() % 10`** to give us numbers from 0-9. If we want numbers from 1-10 we can now just scale up by adding one. The final result is:

```
cout << (rand() % 10) + 1 << endl;
```

Using srand()

If you run this program many times, you'll quickly notice that you end up with the same sequence of random numbers each time. This is because these numbers are not truly random, but pseudorandom. Repeat calls to rand merely return numbers from some sequence of numbers that appears to be random. Each time we call rand, we get the next number in the sequence.

If we want to get a different sequence of numbers for each execution, we need to go through a process of ***randomizing.*** Randomizing is **"seeding"** the random number sequence, so we start in a different place. The function that does this is `srand()` which takes an integer as the seed:

```
void srand(int seed);
```

It is important to only invoke the srand() call ONCE at the beginning of the program. There is no need for repeat calls to seed the random number generator (in fact, it will make your number less evenly distributed).

A commonly used technique is to seed the random number generator using the clock. The `time()` function will return the computer's time. On transformer, this is expressed in terms of the number of seconds that have elapsed since Jan 1, 1970 (the Epoch). The function `time(nullptrptr)` will return the number of seconds elapsed in computer time.

```
#include <ctime>
srand(time(nullptr));
cout << (rand() % 10) + 1;
```

This produces different random values each time the program is run.

Extending rand() to 32bits on Windows (not on macOS)

If you use **`rand()`** to generate random numbers, it may good enough because of its range of the value it produces. **`rand()`** returns a random number between 0 and **`RAND_MAX`** which is defined in **`stdlib.h`** or **`cstdlib`**. **`RAND_MAX`** is surprisingly small (guaranteed at least **`32767`**, **`0x7FFF`**), neither 1M nor 1B) and also depends on the machine.

Quite often we need to generate a wider range of random numbers. Then we can extend it as shown below:

1) Take two random numbers `a, b`
2) Calculate `a * (RAND_MAX + 1) + b`

This will generate random values in the range `[0, (RAND_MAX+1)*(RAND_MAX+1))` on PC/Windows. This algorithm is coded in **`nowic/src/rand.cpp`** and available through **`now/include/rand.h`** and **`nowic/lib/librand.a.`**

```
/// It returns an extended random number of which the range is from 0
/// to (RAND_MAX + 1)^2 - 1. // We do this since rand() returns too
/// small range [0..RAND_MAX) where RAND_MAX is usually defined as
/// 32767 in cstdlib. Refer to the following link for details
/// https://stackoverflow.com/questions/9775313/extend-rand-max-range
/// this should work for PC/Windows and OSX as well.

unsigned long rand_extended();
```

RAND_MAX on macOS is big enough, we don't need this kind of patch at all.