CSI 201
randomness and modulus!

1. Goals for today: learning randomness

2. Everyone, write 10 random numbers on a paper. Now, everyone gets a different number that represents which seed they are. If I choose seed 1 and ask for a random number, I'll get the first number from that seed. If I ask for another random number, I'll get the second number from that seed. If I restart my program and choose seed 1 again, I'll get the same list of numbers even though I'm asking for random numbers. This is called pseudo-randomness. Hopefully this helps you understand the concept.

3. So how do we get randomness in code? We need two pieces of functionality and a few lines of code. The first piece is `srand`. It let's the library know which seed to pick random numbers from (our library for this is cstdlib). Secondly, we'll need `rand`. It will give us a random number non-negative integer (0 and above). Let's tackle these pieces.

4. `#include <cstdlib>` – To use `srand` and `rand` in your code, you'll need to make sure to `#include <cstdlib>` This the same as the includes we needed for cout/cin

5. `void srand(int);` – In any code where you want randomness, you should first seed the random number generator (RNG). You do this using `srand`. So if you want to select the seed 0 you might have code: `srand(0);`. You can even allow the user to type in the seed number like below. Later we'll see how to get a different seed every time we run the program.

   ```
   int num = 0;
   cin >> num;
   srand(num);
   ```

6. `int rand();` – To get a random number, you just use `rand()`. You can assign that integer to a variable and use it as you like! Technically, `rand()` gives us a number from 0 to `RAND_MAX`. `RAND_MAX` can be pretty low. Some systems have a `RAND_MAX` of 32767 (although that is pretty low), but it might be more or even less depending on your architecture. We'll briefly look at better randomness later in this document. In any case, you might want to use `RAND_MAX` to get a random number between 0 and 1 (not just 0 and 1 but any number, including decimals, between 0 and 1, like 0.5, 0.2, or 0.23425). A sample seed, rand, and some uses are in the next code sample:

7.
```
#include <vector>
#include <iostream>
#include <cstdlib> //need this for rand and srand
using namespace std;
int main() {
  int seed_num; //hey user give me a seed
  cin >> seed_num;
  srand(seed_num);
  cout << "A random number: " << rand() << endl;
  double the_rand = rand(); //implicit type cast from int to double
  double zero_to_one = the_rand / RAND_MAX;
  cout << "Random decimal 0-1: "
       << zero_to_one << endl;
}
```

8. Having a random integer is useful, but generally we want a random number inside a certain range. Maybe it is a random number that represents a normal die roll (random numbers from 1 to 6). To do this, we use the modulus operator. If the total number of random possible values that we want is 6, then we can take a random integer and modulus by 6. But if we take any random integer and mod it by 6, we get a number from 0 to 5. So, to shift this range from 0 to 5, we add 1. An example is below in code snippet form.

```
//srand and other things happen earlier
int random_integer = rand();
int random_d6 = (rand()%6) + 1;
int random_d8 = (rand()%8) + 1; //random from 1 to 8.
//we can get any range of numbers, say 10-12 if we want.
//rand()%3 gives a number 0, 1 or 2.
//We add 10 to get 10, 11 or 12
int random_10_to_12 = rand()%3 + 10;
```

9. Can you write some code to get a random number from 1 to 100?

10. Try writing some code to get a random number from 1 to 3. But, make this code such that the 1 is 3 times as likely as the 3 and the two is twice as likely as a 3. You can get this affect in some games with a 6-sided die who's faces are 1,1,1,2,2,3 instead of 1,2,3,4,5,6.

11. Putting some things together! Try filling a vector of 20 entries with randomness.

12. Typing in a seed every time could give a user an advantage in some tasks. Imagine typing in the seed value for a quiz such that you get exactly the questions you desire! Or imagine playing a card game with a friend where they know exactly where all the cards lie and you do not. So, there is a mechanism to get a new seed every time you run a program. Remember our discussion of the Unix epoch time? It is the number of seconds since January 1, 1970. There's a utility in `#include<ctime>` that will give you the Unix epoch time. Usually as an integer. That function is called time. The way we'll use it, is by typing: `time(0)`. This will give us the number of seconds. And every time we run our program, that number of seconds will be different. And thus, we'll get a different seed! Unless we manage to run the program twice in the same second. Try out this simple program below a few times.

13. It requires `#include<ctime>` at the top of your program. And then you simply seed your program with `time(0)`. Here is an example (definitely run it a few times):

```
#include <cstdlib>
#include <ctime>
#include <iostream>
using namespace std;
int main() {
  srand(time(0));
  cout << rand() << endl;
  cout << rand() << endl;
}
```

14. Final considerations! It is uncommon to use rand and srand for anything too serious. They're not likely thread safe, and if you end up in an architecture with a small max, the distribution only involves 32k possible values. C++11 has an entirely different built in random number generator that allows you to pick an engine and a distribution. Technically, using modulus, you're not likely to get a truly or uniformly random distribution in either environment. But they're easy to learn to start and they're okay to use for making your first programming game. In addition, there are a lot of discussions and measurements about what "truly random" and "randomly distributed" might actually mean in a finite use case. Nevertheless, it is generally agreed that

using rand/srand/modulus is definitely not. So, if you're interested, take a dive into C++11's `#include<random>` and do some reading about all it has to offer.