CS111

Introduction to Computing Science

Recap

Previously

- Standard input and output
- Data types
- Ifs and it-else
- Nested and multiple ifs
- Loops
- Nested Loops

Today:

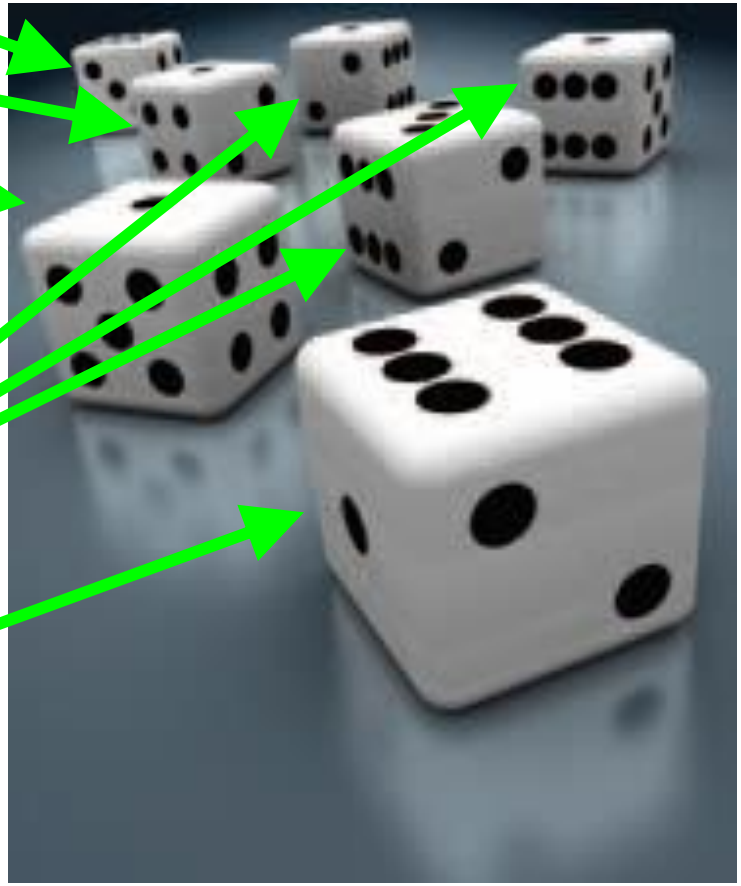
- Random numbers

Random Numbers and Simulations

A die toss

another die toss

and another die toss



several tosses of dice

one more die is tossed

was that an English lesson?

Simulations

A *simulation program* uses the computer to simulate an activity in the real world (or in an imaginary one).

Simulations

- Simulations are commonly used for
  - Predicting climate change
  - Analyzing traffic
  - Picking stocks
  - Many other applications in science and business

Randomness for Reality (Simulating)

- Programmers must model the “real world” at times.
- Consider the problem of modeling customers arriving at a store.

  Do we know the rate?

  Does anyone?

  How about the shopkeeper!

Randomness for Reality (Simulating)

Ask the shopkeeper:

It's about every five minutes
…or so…
…give or a take a couple…
…or three…

…but on certain Tuesdays…

Randomness for Reality (Simulating)

To accurately model customer traffic, you want to take that random fluctuation into account.

How?

The `rand` Function

The C++ library has a random number generator:

```
rand()
```

The `rand` Function

`rand` is defined in the **`cstdlib`** header

Calling **`rand`** yields a random integer between 0 and **`RAND_MAX`**

(The value of **`RAND_MAX`** is implementation dependent)

The `rand` Function

Calling **`rand`** again yields a different random integer

Very, very, very rarely it might be the same random integer again.

(That's OK. In the real world this happens.)

Modeling Using the `rand` Function

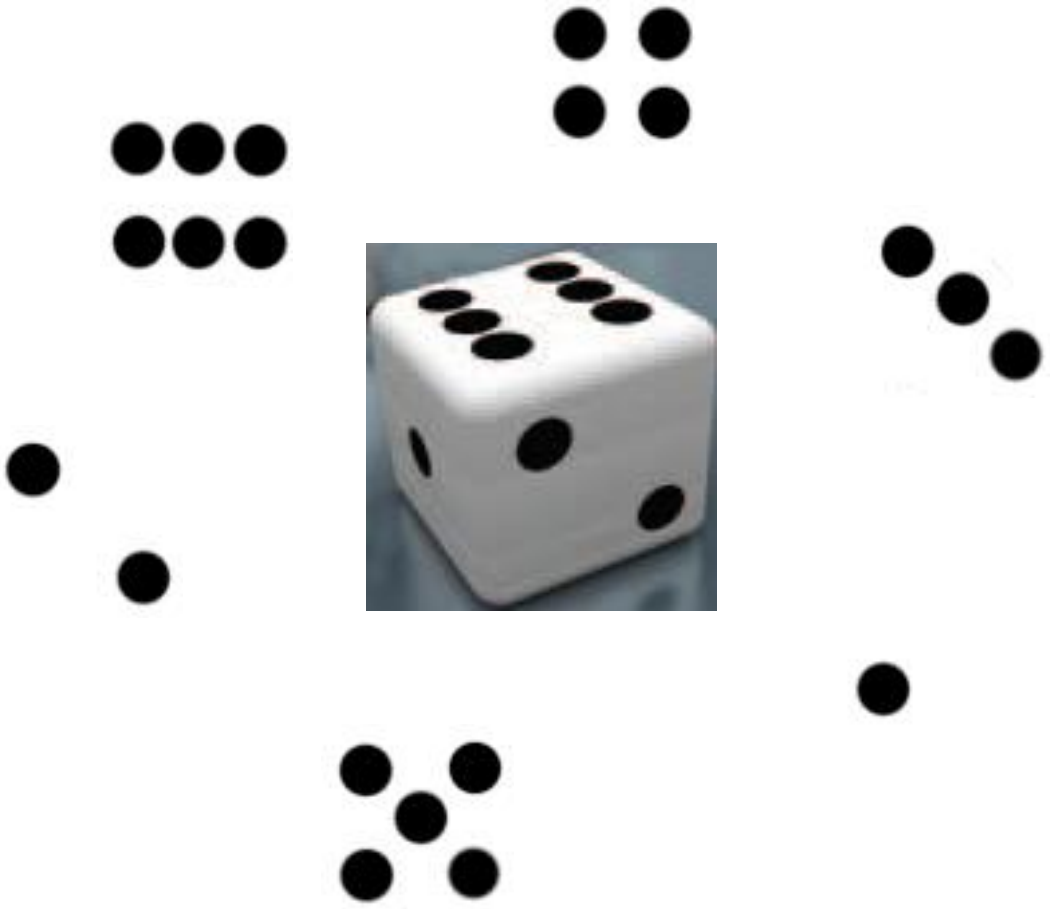
Let's model a pair of dice,

Modeling Using the `rand` Function

one die at a time.

Modeling Using the `rand` Function

What are the numbers on one die?



Modeling Using the `rand` Function

Numbers we can work with please!

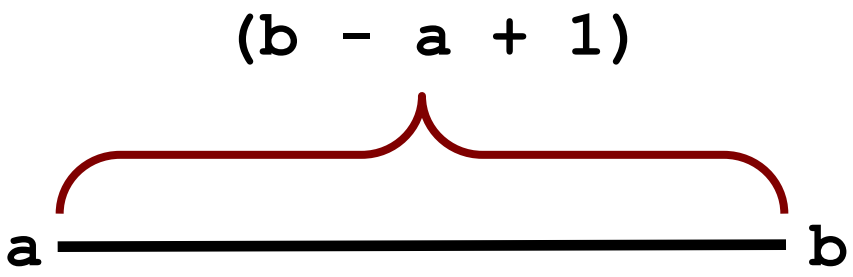
Modeling Using the `rand` Function

What are the bounds of the range of numbers on one die?

1 and 6 (inclusive)

We want a value randomly between those endpoints (inclusively)

Modeling Using the `rand` Function

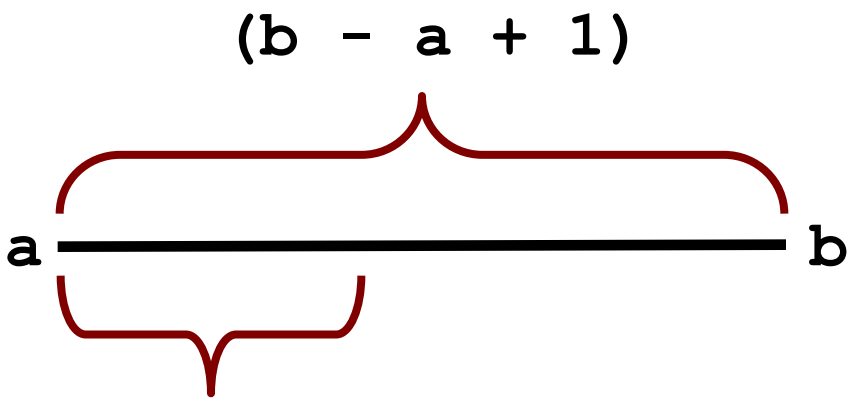


Given two endpoints,
a and **b**, recall there are

(b - a + 1)

values between **a** and **b**,
(including the bounds themselves).

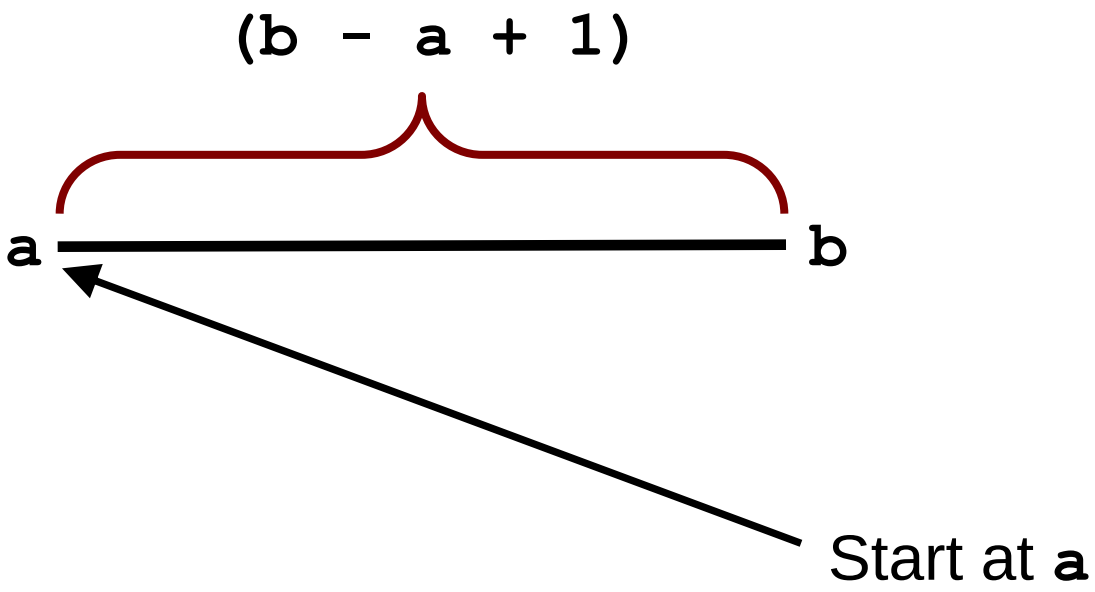
Modeling Using the `rand` Function



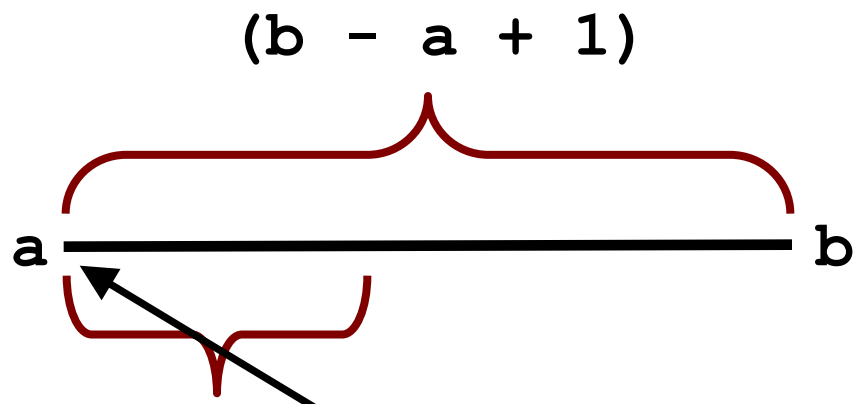

```
rand() % (b - a + 1)
```

Obtain a random value
between **0** and **b - a**
by using the **rand()** function

Modeling Using the `rand` Function



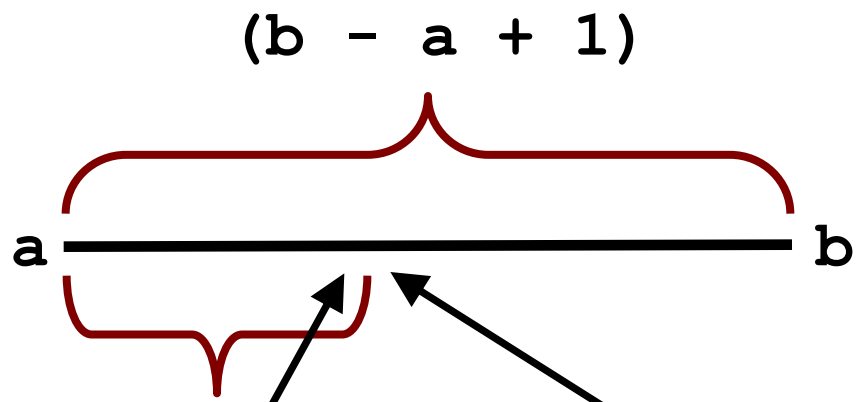
Modeling Using the `rand` Function



Add that random value to **a** and you have:

```
int d = rand() % (b - a + 1) + a;
```

Modeling Using the `rand` Function




```
rand() % (b - a + 1)
```

a random value in the range.

```
int d = rand() % (b - a + 1) + a;
```

Modeling Using the `rand` Function

Using 1 and 6 as the bounds
and
modeling for two dice,
running for 10 tries,

we have:

Modeling Using the `rand` Function

```
#include <iostream>
#include <string>
#include <cstdlib>
#include <ctime>
using namespace std;

int main()
{
   srand(time(0));

   for (i = 1; i <= 10; i++)
   {
      int d1 = rand() % 6 + 1;
      int d2 = rand() % 6 + 1;
      cout << d1 << " " << d2 << endl;
   }
   cout << endl;
   return 0;
}
```

One of many different Program Runs:

```
5 1
2 1
1 2
5 1
1 2
6 4
4 4
6 1
6 3
5 2
```

Bob’s program

- Adding random choice to Bob’s program.

Alice’s program

- Adding random guesses to Alices’s program.

The `rand` Function

`rand` picks from a very long sequence of numbers that don't repeat for a long time.

But they do eventually repeat.

These sorts of "random" numbers are often called *pseudorandom numbers*.

The `rand` Function

`rand` uses only one pseudorandom number sequence

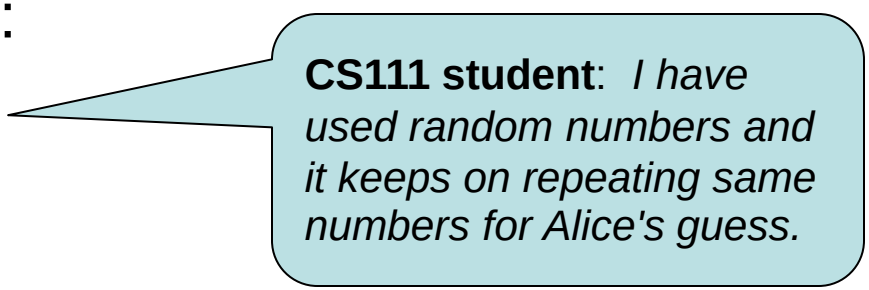
and it always starts from the same place.

Oh dear

The `rand` Function

When you run your program again on another day, the call to **`rand`** will start with:

*the **same** random number!*



Is it very “real world” to use the same sequence over and over?

No, but it’s really nice for testing purposes.

but…

Seeding the `rand` Function

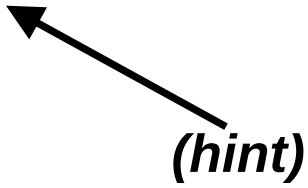
You can “seed” the random generator to indicate where it should start in the pseudorandom sequence

Calling **`srand`** sets where **`rand`** starts

`srand` is defined in the **`cstdlib`** header

Seeding the `rand` Function

But what value would be different every *time* you run your program?



How about the time?

Seeding the `rand` Function

You can obtain the system time.

Calling **`time(0)`** gets the current time

Note the zero. It is required.

`time` is defined in the **`time`** header

Seeding the `rand` Function

Calling **`srand`** sets where **`rand`** starts.

Calling **`time(0)`** gets the current time.

So, to set up for "really, really random" random numbers on each program run:

```
srand(time(0)); // seed rand()
```

(Well, as "really random" as we can hope for.)

Quiz

- What gives you a random number?

Quiz

- What gives you a random number between a and b?

Quiz

- What do you need to do to get a different random number each time you run?