(pseudo)random number generators

- we often want to give our programs a level of unpredictability, such as when playing a game against the computer
- this is achieved by having the computer generate seemingly random values, then taking different action based on the value
- the generators are themselves computer programs, so their actions and the values produced are, in fact, predictable *with sufficient background information*, but the values *appear* random to the people using them, hence the “pseudo-”random designation
- while I say “random” in future slides, keep in mind that it really is pseudo-random

rand() in C++

- one random number generator available to us the the rand() function from the cstdlib library
- when called, rand returns an apparently random non-negative long integer, e.g.
- x = rand(); // x now holds some random integer
- of course, that means the value could be anything from 0 to 9223372036854775807 (if I typed that right)
- usually we want a random number in a smaller range...

seeding the generator

- every pseudorandom sequence is based off of a different starting “seed” value used to initialize the generator
- if we don't seed the generator, or if we always use the same seed, then the program will always generate the same sequence
- an internal time is often used as a way to seed the generator, so that it will be different every time we run the program
- we only need to seed the generator once at the start of the program, then we can call the random number routine as often as desired

srand and time

- srand(seed) is the function to seed rand's generator
- time(NULL) is a function call to get the internal time

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

int main()
{
    // seed the generator
    srand(time(NULL));
    ... now for the rest of the program we can use rand as often as we want ...
```

rand() and modulo

- suppose we want a random number in the range 0..N, e.g.
  - generate random value, r, from 0 to 3
  - if r is 0 then the AI moves north
  - else if r is 1 then the AI moves west,
  - etc
- the easiest way to get a value in the desired range is to call rand() then get the remainder after dividing by (N+1)

```
int rval;
rval = rand() % 5; // gives random int from 0 to 4
rval = rand() % 101; // gives random int from 0 to 100
```

Example: flip a coin

```
// get user to guess result of a coin flip,
// will use 0 internally for tails, 1 for heads

cout << “Pick H for heads or T for tails” << endl;
char pick;
cin >> pick;

int coinflip = rand() % 2;
if ((pick == 'H') && (coinflip == 1)) {
   cout << “Correct, heads!” << endl;
} else if ((pick == 'T') && (coinflip == 0)) {
   cout << “Correct, tails!” << endl;
} else {
   cout << “Wrong!” << endl;
}
```

Example: pick a card

- will represent a card using two integer variables

```
// one int represents the suit
// (0=hearts, 1=spades, 2=diamonds, 3=clubs)
// one int represents the rank
// (1=ace, 2=2, ..., 10=10, 11=jack, 12=queen, 13=king)
// now generate a random card
int suit = rand() % 4;
int rank = 1 + rand() % 13
// added 1 to get rank in the range 1..13 instead of 0..12
```

random(M, N)

- suppose we want a function that returns a random integer in the range M..N
- there are (1+N-M) possible values in the range, so we can use M + (rand() % (1+N-M)) to get our desired value ... verifying this is left as an exercise to the reader :)

```
// return a random integer in the range M..N
long random(long M, long N)
{
   return M + (rand() % (1 + N - M));
}
```