601.220 Intermediate Programming

Random numbers

Plan for today

- Pseudo-random integers in C

Pseudo-random integers in C

- `rand()` generates (pseudo) random integers between 0 and `RAND_MAX`
  - distribution is uniform: each value in range is equally likely to be generated
- the pseudo random sequence of integers is based on a *seed*
  - different seed $\rightarrow$ different sequence of pseudo-random values
- `srand( unsigned int )` sets the seed value
- if `srand()` is not called, by default, it uses seed `1` (as if `srand(1)` called at the beginning of the program)
- use `srand(time(0))` to generate time dependent random integers (`time.h` is required)

Pseudo-random integers in C

```c
// random.c:
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
    for (int i = 0; i < 5; ++i)
        printf(" %d ", rand()); // print 5 random integers w/o calling srand()
    printf("\n");
    srand(time(0)); // Set seed to current time
    for (int i = 0; i < 5; ++i)
        printf(" %d ", rand()); // print another 5 random integers
    printf("\n");
    srand(1); // Set seed back to 1
    for (int i = 0; i < 5; ++i)
        printf(" %d ", rand()); // print another 5 random integers
    printf("\n");
    return 0;
}
```

```
$ gcc -std=c99 -pedantic -Wall -Wextra -c random.c
$ gcc -o random random.o
$ ./random
 1804289383  846930886  1681692777  1714636915  1957747793
 1289180750  1670867387  1193368706  1314336278  45542572
 1804289383  846930886  1681692777  1714636915  1957747793
```

Generating pseudo-random integers in a specific range

The modulus (%) operator is useful for constraining the range of values generated by `rand()`.

Examples:

Code	Range of values (inclusive)
`rand()`	0 to `RAND_MAX`
`rand() % 100`	0 to 99
`rand() % 101`	0 to 100
`(rand() % 100) - 50`	-50 to 49
`(rand() % 101) - 50`	-50 to 50

Generating pseudo-random floating point values

One way to generate pseudo-random floating-point values is to map a range of integers onto real numbers.

Examples:

Code	Range of values (inclusive)
`((rand() % 100000) / 100000.0)`	0.0 to 0.99999
`((rand() % 100001) / 100000.0)`	0.0 to 1.0

Increasing the size of the range improves the "granularity" of the values generated. Finest granularity for generating values between 0 and 1 (inclusive): `rand() / (double)(RAND_MAX - 1)`.