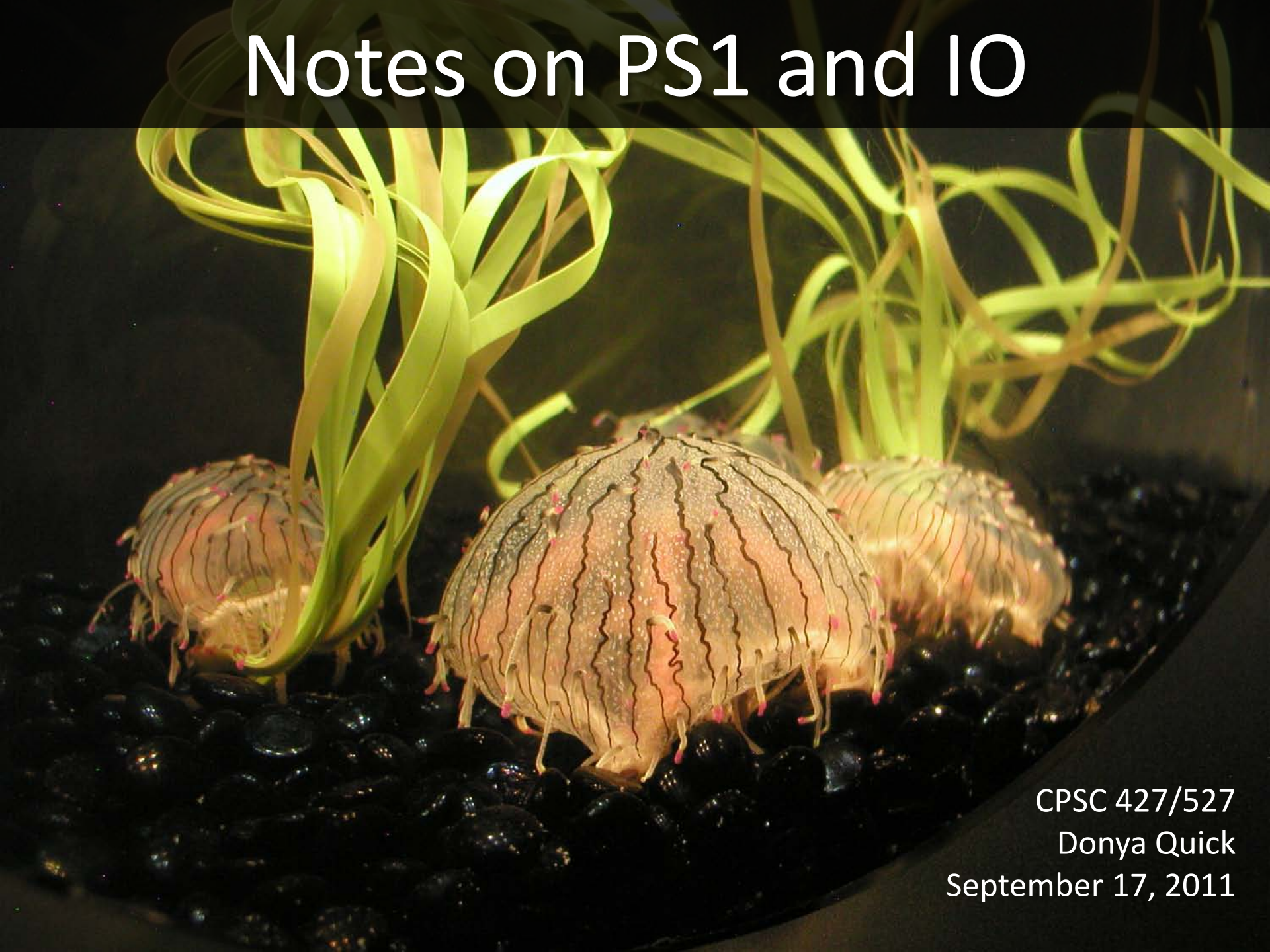
Notes on PS1 and IO



CPSC 427/527
Donya Quick
September 17, 2011

Outline

- Assignment & e-mail mechanics
- General feedback on PS1
- IO example

Turning in Assignments

- For PS1: DO NOT RESUBMIT. The information below does not apply to PS1.
- For all future assignments, PS2 and onward:
  1. **<u>All</u> electronic material should be submitted via the zoo "submit" script.**
     - Do NOT use the classes*v2 drop boxes or my e-mail to submit assignments.
  2. If you submit any physical papers with only a paperclip, please write your name on all pages.

Nicknames & NetIDs

I often get e-mails and files that look similar to the fictitious examples below.

From: phillcollins@gmail.com
Subject: compiling foo.cpp

How does foo.cpp relate to problem set #243? Also, did you get my last homework submission?

----Rob

If "Rob" is a nickname, it will not appear on the classes*v2 roster.

```
/*
* Bar.cpp
* Created by jrs527049
*/

public int main() {
…
}
```

A NetID is much better than no identification at all, but it is still time-consuming to match to a name.

Nicknames & NetIDs

I need more information to easily know who you are!

From: phillcollins@gmail.com
Subject: compiling foo.cpp

How does foo.cpp relate to problem set #243? Also, did you get my last homework submission?

----Rob (John Smith)

```
/*
* Bar.cpp
* Created by John Smith (jrs527049)
*/

public int main() {
…
}
```

Common Feedback on PS1 Code (1)

- **Put your name in all documents you create!!!**

  Your name = your name as it appears on the roster

  From now on, no name = points deducted

- Put your name in all documents you edit.
  - Always keep the original author's name.
  - Ex: "Created by John Smith" should become "Created by John Smith, modified by [YOU]" if you change anything in the file.

Common Feedback on PS1 Code (2)

- Comment your code appropriately.
  - This is *not* optional, even on small assignments.
    - It helps demonstrate to me that you understand your code.
  - Make sure final comments are consistent with your code.
    - Especially important when modifying existing code.
  - Do not leave commented-out blocks of code in your final version.
    - This includes heavy usage of `cout` statements used for debugging. It is better to use a debugging class.
    - Exceptions for turning in partial work if you want to show what you did but need to disable it for compiling purposes.

Well-Documented Code(1)

```
/*
* File name
* Created by [AUTHOR]
* Last modified [DATE]
* [Description of what this class is meant to do]
*/

...

/*
* Function's purpose
* Description of arguments
* Preconditions, if any
* Postconditions, if any
*/
public void myMethod(...) {
    ...
}
```

Well-Documented Code (2)

```
/* function description */
public void myMethod(...) {
    ...
    // description of what loop does
    while (...) {
       [lengthy loop]
    }

    ...
    // description of lengthy test series
    if (...) {
       ...
    } else if (...) {
       ...
    } else { ... }
}
```

Common Feedback on PS1 (3)

- Pay attention to details. For example, there were three written components:
  1. Highlighting/annotating existing code.
  2. Discussing two specific OO topics.
  3. A brief report on the coding portion.

- Follow the submission instructions. You will lose points if you do not submit required files. For example:

  "You should submit the following items: […]
  3. One or more test files **and corresponding output files** […]."

Common feedback on PS1 (4)

- Make sure your code compiles with the makefile you provide.
  - Submissions that don't compile easily will get automatic zeros on relevant criteria.*
- How to check that your code compiles:
  - Please call your file "makefile" for simplicity.
  - Go to the directory containing the file called `makefile` and run the command `make`
- You MUST tell me how to compile your code if it involves something other than running `make`!
  - Should be described in your report.txt

* This can be turned into partial credit later (next slide)

Common feedback on PS1 (5)

- What to do if you lost points on compilation:
  - Come to my office hours.
    - Tuesday: 4-5pm
    - Wednesday: 1:30-3:30pm
    - If you have a class or other regular mandatory meeting during those times, e-mail me to set up another time.
  - If you can make your submitted files compile, I will re-grade applicable test cases.

PS1 Solution

(Viewed in Eclipse)

IO Examples

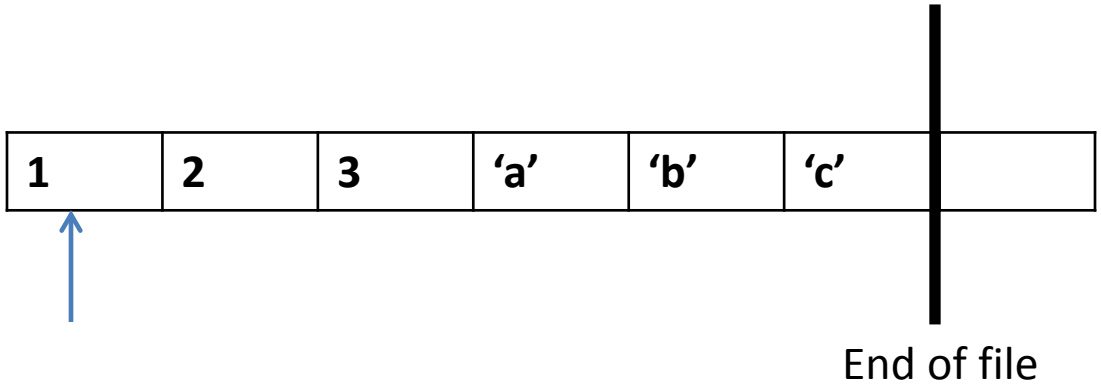
IO Example 1

- How eof gets set.

eofbit = 0
failbit = 0
badbit = 0

good() => true
fail() => false
eof() => false

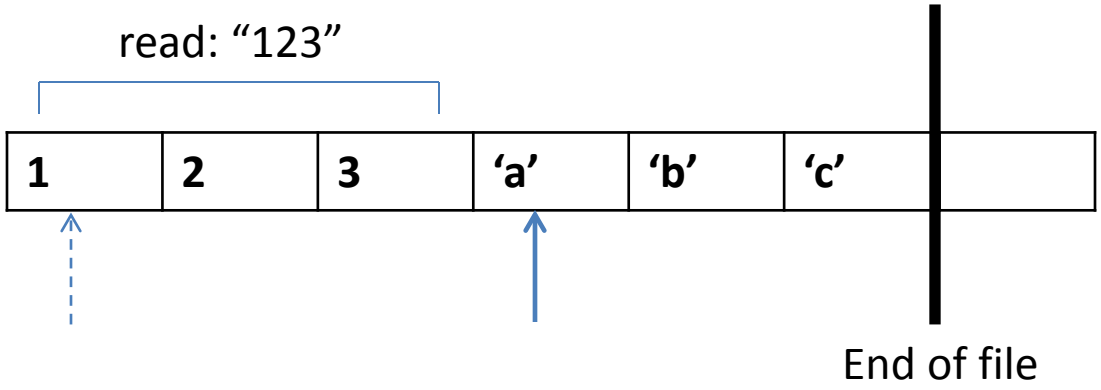
```
ifstream infile( filename );
int x;
string y;
char ch;
…
infile >> x;
ch = infile.peek();
infile >> y;
ch = infile.peek();
```



eofbit = 0
failbit = 0
badbit = 0

good() => true
fail() => false
eof() => false

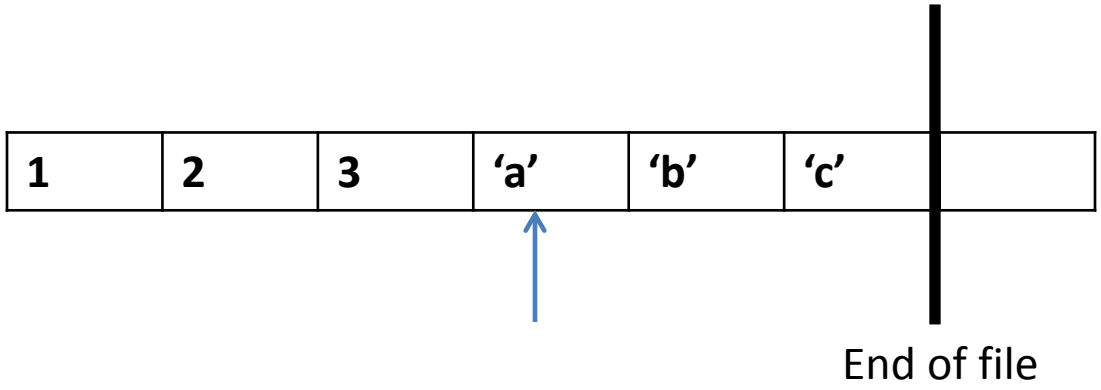
```
ifstream infile( filename );
int x;
string y;
char ch;
…
infile >> x;
ch = infile.peek();
infile >> y;
ch = infile.peek();
```



eofbit = 0
failbit = 0
badbit = 0

good() => true
fail() => false
eof() => false

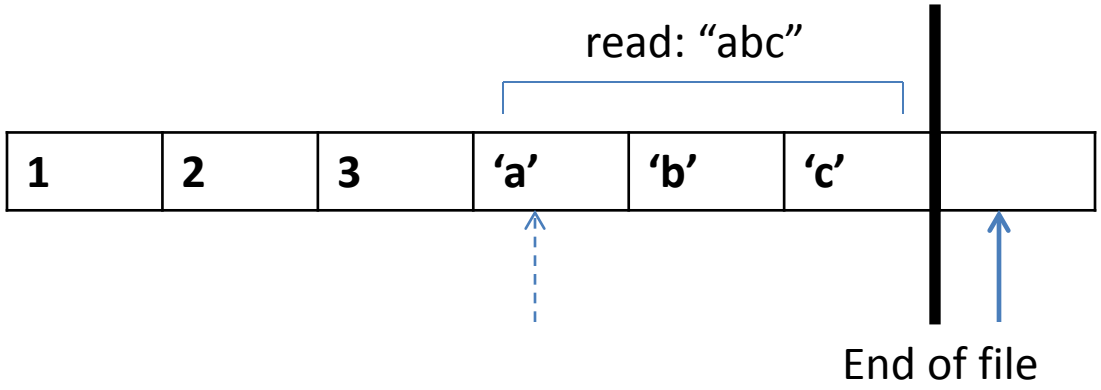
```
ifstream infile( filename );
int x;
string y;
char ch;
…
infile >> x;
ch = infile.peek();
infile >> y;
ch = infile.peek();
```



eofbit = 0
failbit = 0
badbit = 0

good() => true
fail() => false
eof() => false

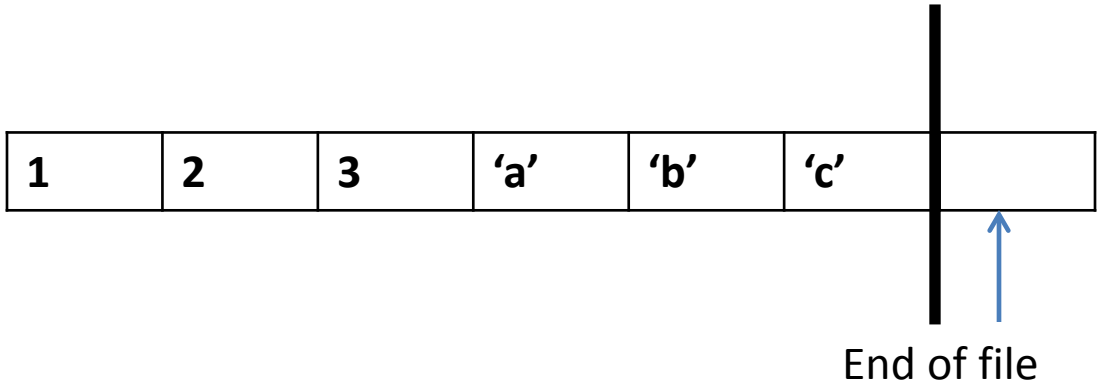
```
ifstream infile( filename );
int x;
string y;
char ch;
…
infile >> x;
ch = infile.peek();
infile >> y;
ch = infile.peek();
```



eofbit = 1
failbit = 0
badbit = 0

good() => false
fail() => false
eof() => true

```
ifstream infile( filename );
int x;
string y;
char ch;
…
infile >> x;
ch = infile.peek();
infile >> y;
ch = infile.peek();
```



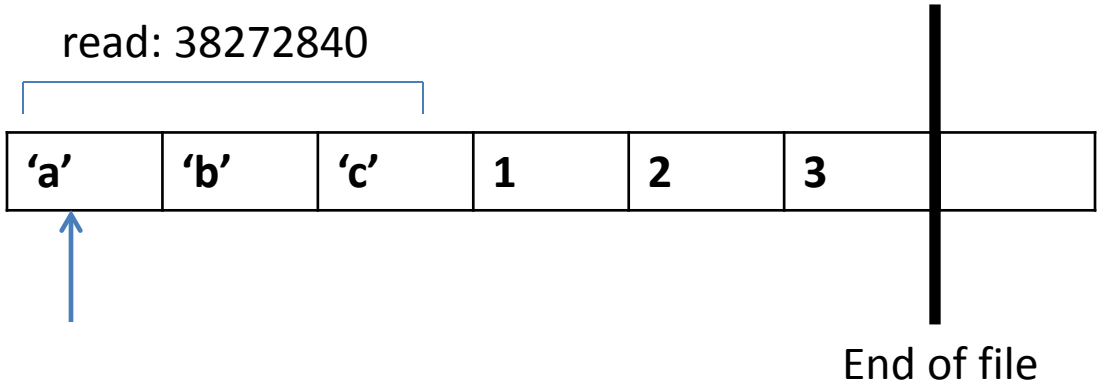
IO Example 2

- Reading bad data

eofbit = 0
failbit = 0
badbit = 0

good() => true
fail() => false
eof() => false

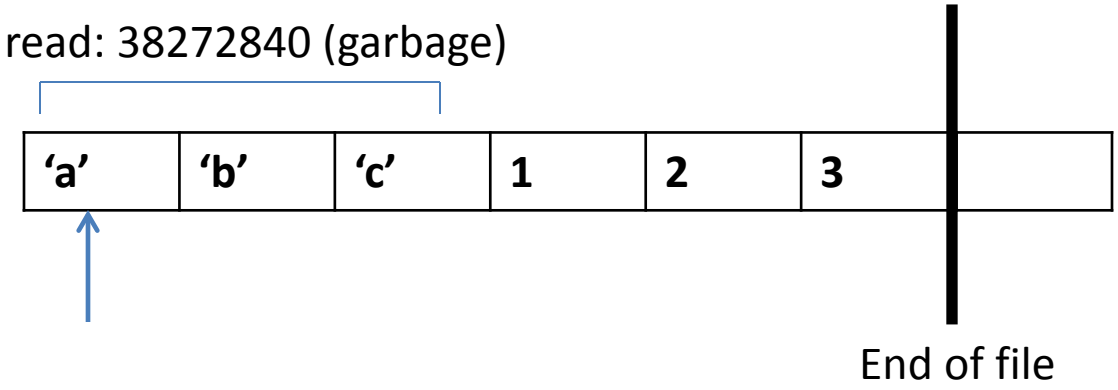
```
ifstream infile( filename );
int x;
string y;
char ch;
…
infile >> x;
ch = infile.peek();
infile >> y;
ch = infile.peek();
```



eofbit = 0
failbit = 2
badbit = 0

good() => false
fail() => true
eof() => false

```
ifstream infile( filename );
int x;
string y;
char ch;
…
infile >> x;
ch = infile.peek();
infile >> y;
ch = infile.peek();
```



Problem Set 2

Random number generation and simulations

Pseudorandom number generators

You will need to generate random numbers in this assignment.
A few remarks on random number generation are in order.

- Pseudorandom numbers are *not* random. They are *predictable*. This is both an asset and a curse.
- Since they are predictable, a simulation run can be repeated to obtain the same results, particularly helpful during debugging.
- Since they are not random, they may have statistical properties that differ from true random numbers.
- "Good" pseudorandom numbers should pass common statistical tests for randomness.

Random numbers in C++

- `rand()` is standard random number generator in C and C++.
- `rand()` implementation on current Linux systems is good but not on some other systems.
- Newer and better random number generators might be preferable for real-world applications.

`rand()` and `srand()`

Basic properties

- `int rand(void)` generates next number in sequence using hidden internal state.
- Not thread safe.
- `void srand(unsigned int seed)` initializes the state.
- Seed defaults to 1 if `srand()` not called.
- `rand()` returns an `int` in the range `[0...RAND_MAX]`.
- Must `#include <cstdlib>`
- `RAND_MAX` is typically the largest positive number that can be represented by an `int`, e.g., $2^{31} - 1$.
- The result from `rand()` is rarely useful without further processing.

Generating uniform distribution over a discrete interval

To generate a uniformly distributed number $u \in \{0, 1, \ldots, n-1\}$:

- **Naive way:** `u = rand()%n`.
  Problem: Result not uniformly distributed unless $n \mid$ `RAND_MAX`.
- **Better way:**

```
int RandomUniform(int n) {
  int top = ((((RAND_MAX - n) + 1) / n) * n - 1) + n;
  int r;
  do {
    r = rand();
  } while (r > top);
  return (r % n);
}
```

Generating random doubles

To generate a double in the semi-open interval $[0 \ldots 1)$:
`(double) rand() / ( (double)(RAND_MAX) + 1.0 )`

- Without `+ 1.0`, result is in the closed interval $[0 \ldots 1]$.
- `(double) rand() / ( RAND_MAX + 1 )`
  might fail because of integer overflow.

Alternate method for generating uniform distribution over a discrete interval

To generate a uniformly distributed number $u \in \{0, 1, \ldots, n-1\}$:

1. `#include <cmath>`.
2. Generate a uniformly distributed random double `u` in $[0 \ldots 1)$.
3. Compute `trunc(n*u)`.

Question: Is this truly uniform over $\{0, 1, \ldots, n-1\}$?

Generating exponential distribution

[Not needed for PS2 but useful to know.]

To generate a double according to the exponential distribution with parameter `lambda`:

1. `#include <cmath>`.
2. Generate a uniformly distributed random double `u` in $[0 \ldots 1)$.
3. Compute `-log(1.0-u)/lambda`.

Note: `log(0.0)` is undefined. Will return a special value that prints as `-inf`.