

CPSC 427a: Object-Oriented Programming

Michael J. Fischer

Lecture 7
September 23, 2010

Remarks on Problem Sets

Problem Set 1

Problem Set 2

BarGraph Demo

`graph.hpp`

Remarks on Problem Sets

Problem Set 1

Seth Hamman

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 <stdlib>`
- ▶ `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 $u \in \{0, 1, \dots, n-1\}$:

- ▶ **Naive way:** `u = rand()%n`.

Problem: Result not uniformly distributed unless $n \mid \text{RAND_MAX}$.

- ▶ **Better way:** See problem set.

```
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 $[0 \dots 1)$:

```
(double) rand() / ( (double)RAND_MAX + 1.0 )
```

- ▶ Without `+ 1.0`, result is in the closed interval $[0 \dots 1]$.
- ▶ `(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 $u \in \{0, 1, \dots, n - 1\}$:

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

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

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 $[0 \dots 1)$.
3. Compute `-log(1.0-u)/lambda`.

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

Bar Graph Demo

We look at the Bar Graph demo from Chapter 8 of the textbook.

graph.hpp

```
class Graph {
private:
    Row* bar[BARS]; // List of bars (aggregation)
    void insert( char* name, int score );
public:
    Graph ( istream& infile );
    ~Graph();
    ostream& print ( ostream& out );
    // Static functions are called without a class instance
    static void instructions() {
        cout << "Put input files in same directory "
              "as the executable code.\n";
    }
};

inline ostream& operator<<( ostream& out, Graph& G) {
    return G.print( out );
}
```