Random Numbers

- Random Numbers
- Linear Conguential Generators
- Random Number Library

❖ Random Numbers

How can a computer generate a random number?

- in theory, it can't, because it behaves deterministically

Software can only produce pseudo random numbers.

- a pseudo random number is one that is predictable
- but varies enough that it appears unpredictable

A generator of pseudo-random numbers might be "good enough"

- if it generates values uniformly within a range
- and it is not obvious to an observer what the next number will be

❖ ... Random Numbers

Ideally, we want a function `random()`

- that produces a random number each time we call it
- the next number is unrelated to the previous number

Using it in the code ...

```
int freq[10] = {0,0,0,0,0,0,0,0,0,0};
for (i = 0; i < 100000; i++) {
   int n = random() % 10;
   freq[n]++;
}
```

... after which each freq[i] should contain the same value (10000)

❖ Linear Conguential Generators

The most widely-used pseudo random number technique:

- Linear Congruential Generator (LCG)

LCG uses a *recurrence* relation:

- $X_{n+1} = (a \cdot X_n + c)\ \mathit{mod}\ m$, where:
  - m is the "modulus"
  - a, $0 < a < m$ is the "multiplier"
  - c, $0 \le c \le m$ is the "increment"
  - X_0 is the "seed"
- if c=0, it is called a multiplicative congruential generator

Note: requires X_n to be saved between calls to the LCG function.

❖ ... Linear Conguential Generators

Typical implementation of an LCG

```
#define a ???
#define c ???
#define m ???

unsigned int random()
{
   static unsigned int X;
   X = (a * X + c) % m;
   return X;
}
```

It is possible to omit **`m`** if **`a`** is large and we ignore overflows

- effectively treat it as **`mod 2`**^{32}, so generate values 0..2^{32}-1

❖ ... Linear Conguential Generators

Clearly, we need to be careful in choice of **a**, **c** and **m**

Consider the case where **a**=11, **m**=31, **c**=0, $\mathbf{X_0=1}$

An LCG would produce the sequence:

```
11, 28, 29, 9, 6, 4, 13, 19, 23, 5, 24, 16, 21, 14,
30, 20, 3, 2, 22, 25, 27, 18, 12, 8, 26, 7, 15, 10,
17, 1, 11, 28, 29, 9, 6, 4, 13, 19, 23, 5, 24, 16,
21, 14, 30, 20, 3, 2, 22, 25, 27, 18, 12, 8, 26, 7,
15, 10, 17, 1, 11, 28, 29, 9, 6, 4, 13, ...
```

Observations:

- all the integers from 1 to 30 appear, in "random" order
- but the sequence repeats after every 30 numbers

❖ ... Linear Conguential Generators

Now consider the case where **a**=12, **m**=30, **c**=0, $\mathbf{X_0=1}$

AN LCG would produce the sequence:

```
12, 24, 18, 6, 12, 24, 18, 6, 12, 24, 18, 6,
12, 24, 18, 6, 12, 24, 18, 6, 12, 24, 18, 6,
12, 24, 18, 6, 12, 24, 18, 6, ...
```

Does not produce all values in the range 1..30, and repeats with short period

To avoid scenarios like this ...

- use large (relatively) prime values for **a**, **m**, **c**

Note: same initial value for $\mathbf{X_0}$ always produces same sequence

❖ ... Linear Conguential Generators

It is a complex task to pick good numbers

Lewis, Goodman and Miller (1969) suggested

- $X_{n+1} = 7^5 \cdot X_n \bmod (2^{31}-1)$
- note:
  - 7^5 is 16807
  - 2^{31}-1 is 2147483674
  - $X_0 = 0$ is not a good seed value

Most systems use more complex LCG-based algorithms

- see www.mscs.dal.ca/~selinger/random/ for an example

❖ Random Number Library

Two functions are required:

```
srandom(int seed) // sets its argument as the seed (X0)

random() // uses a LCG technique to generate random
         // numbers in the range 0 .. RAND_MAX
```

where the constant **RAND_MAX** is defined in **stdlib.h**

Typically, **RAND_MAX** is large (in CSE, RAND_MAX = 2147483647)

The period (before repetition) is long, approximately $16\cdot((2^{31})-1)$

❖ ... Random Number Library

A problem:

- every time you run a program with the same seed
- you get exactly the same sequence of 'random' numbers

Handy if testing a program and need a repeatable "random" sequence.

But how to use **`srandom()`** to set a different seed each time?

- could use the process ID (see **`man 2 getpid`**)
- could use the current timestamp (see **`man 3 time`**)

Note: default seed is 0, if no call to **`srandom()`**

❖ ... Random Number Library

Random numbers are typically generated in a specific range:

```
int randomRange(int lo, int hi)
{
        int n = random() % (hi-lo+1);
        return n + lo;
}
```

LCG is not good for applications that need very high-quality random numbers

- e.g. applications needing *cryptographically secure* random numbers

However, LCG good enough for day-to-day use.

Produced: 14 Jun 2020