Centrum Wiskunde & Informatica

Laboratory Class Scientific Computing WISM454, 2019

Jan-Willem Buurlage

February 6, 2019

Organization

Expectations

- Learn about *random number generators*, *Monte Carlo methods*, and *genetic algorithms*.
- Develop scientific software using C++.
- Perform numerical experiments.
- Write coherent and concise reports.
- Reason about code performance.

Grading

- Hand-in assignments.
- Two ± 20 page reports containing:
  - Exercise solutions
  - Overview and explanation of code
  - Numerical experiments.
- How the final grade is computed:
  - Assignments: 20%
  - Report I: 40%
  - Report II: 40%
- The focus of the course is on developing your programming skills, and writing good reports.

Topics

- Randomized algorithms and tools
    1. Random number generation
    2. Monte Carlo methods
    3. Genetic algorithms
- Introduction to C++, for developing high-performance code
- Software engineering skills, software architecture, scientific programming

Random number generators

RNGs

- Random number generator (RNG): A means to get uniformly distributed numbers.
- We focus on obtaining numbers in the set:

$$M = \{0, 1, \ldots, m - 1\}.$$

- An RNG can be a physical device, process, or a algorithm.

Pseudo RNGs

- For scientific computing, we value the following properties in our RNG
  - Randomness
  - Reproducibility
  - Efficiency
- We focus on computer based RNGs, which are deterministic.
- Deterministic seems paradoxical, usually called pseudo-RNGs (PRNGs).

Iterations

- Classic PRNGs have the form:

$$x_{i+1} = f(x_i).$$

  The next iterate of a sequence of random numbers produced by PRNGs of this form depend completely on the previous iterate.
- We start the sequence by choosing an initial number x_0. This is called the seed.
- If you use the same seed, you get the same sequence $\implies$ reproducible.

Real numbers

- Say we want numbers not in M but in $[0, 1]$. We can scale:

$$\omega_i \equiv \frac{x_i}{m-1}.$$

- Ex. 2.1: or should we scale in another way?
- The usual strategy is to have an engine that generates integers uniformly. Using these random integers, other distributions can be realized.

Linear congruential RNG

- The linear congruential RNG (LCRNG) has the following form for f:

$$f(x) = (ax + c) \bmod m.$$

- Here, a, c, and m are integer parameters that define the LCRNG.
- Easy to implement, but have some drawbacks.

Example:

- Let us choose: $a = 5, c = 2, m = 6, x_0 = 3$
- $(5 \times 3 + 2) \bmod 6 = 5$
- And then the next element in the sequence is ... again 3
- Although there are m possible output numbers, we can have repeating cycles like here (3, 5, 3, 5, 3, 5, ...).
- (Question: should we include the seed in the sequence?)

Period of LCRNG

Definition

The smallest n such that $x_{i+n} = x_i$ is called the period of the LCRNG. If $n = m$, then full period.

- Ex 2.2: what if $c = 0$?
- Full period means that the LCRNG gives a permutation of M.
- *True* uniform distributions would likely produce the same numbers multiple times without repeating.
- Numbers may become very large. We want to use the maximum m that we can still represent efficiently on the computer.

Binary numbers on computers

- Unsigned integers are typically stored in 32 bits ($=4$ bytes) or 64 bits.

$$x = \sum_{i=0}^{n-1} b_i 2^i.$$

- Some examples:

$$2 = 10_2$$
$$5 = 101_2$$
$$23 = 10111_2$$

- Least significant (right), most significant (left).
- Addition throws away most significant bits (overflow). Arithmetic operations on n-bit integers are like working modulo 2^n!

Negative numbers

- Signed integers. Most significant bit is the sign bit.
- If the sign bit is 0 then the number is positive, if it is 1 then it is negative.
- However, it is done in a smart way called two's complement encoding! Corresponding to the following sequence:

$$\{0, \ldots (2^{n-1} - 1), -2^{n-1}, \ldots, -1\}.$$

Two's complement encoding.

- Signed versus unsigned:

$$(-a)_{\mathsf{s}} \equiv (2^n - a)_{\mathsf{u}}$$

- Note: $2^n - a \equiv (2^n - 1) - a + 1$, so in summary: $-a$: invert all bits of a and add 1.
- Subtraction can then be implemented by addition.

$$\begin{aligned}(x + (-y)_{\mathsf{u}}) \bmod 2^n &= (x + 2^n - 1 - y + 1) \bmod 2^n \\ &= (x - y) \bmod 2^n.\end{aligned}$$

Shrage's trick

- Now that we are a bit familiar with binary representation of numbers on computers, we consider possible issues.

<u>Ex 2.4</u>: $m = 2^b, c \neq 0 \implies$ not random in all bits

- For this reason, we want $m = p$ prime, if $c = 0$:

$$f(x) = ax \bmod m.$$

However, what if ax overflows?

- If we could factorize $m = aq$ then:

$$ax \bmod m = ax \bmod aq = a(x \bmod q).$$

(note that this is always smaller then m)

Shrage's trick (II)

- However, we would like m prime...
- Assume $m = aq + r$ with r small, then (try to prove this for your report):

$$b \equiv a(x \bmod q) - r(x \operatorname{div} q)$$

$$ax \bmod m = \begin{cases} b & \text{if } b \geq 0 \\ b + m & \text{otherwise} \end{cases}$$

  and if $r < q$ all numbers involved are less than m, so we can compute without overflow.
- This is called Shrage's trick

Summary

- An RNG engine generates numbers in the set:

$$M = \{0, 1, \ldots, m-1\}.$$

- For scientific experiments, we want reproducibility, efficiency and randomness.
- LCRNGs are simple generators that can generate pseudo-random sequences.
- Correct implementations require you to be aware of how integers are encoded in your computer.

C++

C++

- Compiled language!
  - *source(s)* → compile → *object file(s)* → link → *single executable*
- Source code is portable, but the executable generally is not (contrary to e.g. Java)
- Language features
  - types, functions, control flow statements
- Standard library
  - containers, IO operations, ...
  - implemented using language features (could build this yourself on top of C++!)

Smallest C++ program

```
int main() { return 0; }
```

- The `main` function is called when the C++ program is executed. One main function across all your source files!
- `int main() {}` is actually also a valid C++ program

Output

```cpp
#include <iostream>

int main() {
    // console out
    // read << as 'put to'
    // std is a namespace
    std::cout << "Hello, world!\n";
}
```

Note: semicolon ;

Types

Every entity has a type, which determines what is valid for that entity. Types are used e.g. to denote the type of the return value of a function (as in main), or its parameters.

```
int square(int x) {
    return x;
}

...

std::cout << square(3) << "\n";
```

Built-in types

There are a number of 'fundamental' (not user-defined) types.

- `bool` (1)
- `char` (1)
- `int` (4)
- `double` (8)

```
int x = 3;
int z = x + 5; // FINE!
bool a = false;
bool b = a + 3; // ERROR!
```

Caveats

```
int b = 7.1; // no error!
float a = 3.0;
float a = 312489012480918240.0;
float a = 3124.0f;
```

Narrowing

Lenient with conversions (narrowing!), can be dangerous. C++11:

```
int b{7.1}; // error!
float a{3.0} // error!
auto a = 12345.0; // a is a double!
```

(I generally use `auto` everywhere, and if necessary annotate on the right).

Some useful operations:

```
x += y; // - * / %
++x;
x++;
```

Constants

```
const auto x = 3;
x = 5; // ERROR!
```

Control flow statements

```
int x = 2;

if (x > 3) {
    f();
} else {
    g();
}

while (x < 3) {
    x += 1;
}
```

For loop

```cpp
// this
for (int i = 0; i < 5; ++i) {
    std::cout << i << "\n";
}
// is equivalent to
int i = 0;
while (i < 5) {
    std::cout << i << "\n";

    ++i;
}
```

Pointers, arrays

```
int xs[6] = {0, 1, 2, 3, 4, 5}; // array of ints
int* ys = nullptr; // pointer to int
int* x = &x[3]; // address of 4th element
int y = *x; // y = contents of x
```

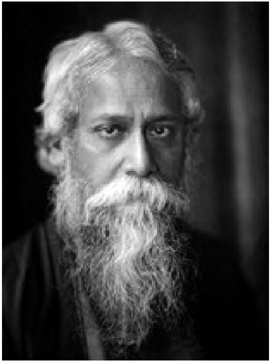
Structures

```
struct lcrng {
    int a;
    int c;
    int m;
};
```

- User defined type!

```
int next(lcrng generator, int x) {
    // ... (generator.a)
}
```

Programming environment



"I have spent many days stringing and unstringing my instrument while the song I came to sing remains unsung."

— *Rabindranath Tagore*

Programming environment (II)

"Sharing is good, and with digital technology sharing is easy."

— *Richard Stallman*, founder of GNU

Minimal C++ programming environment

- *Windows*
  - Notepad++ and CygWin
- *Linux*
  - gedit and GCC
- Your code must be written in *standard C++*, and be buildable with a common cross-platform build tool (more on this in the upcoming weeks).

This week

- Set up programming environment
- Compile and run “Hello, world!”
- Write a simple LCRNG function
- Exercises 2.1, 2.2
- Exercise 2.6: implement and experiment with a number of RNGs
(Note: course website linked to in LNs is outdated)