Using Pseudorandom Number Sequences in C++

Paul Preney, OCT, M.Sc., B.Ed., B.Sc.
preney@uwindsor.ca

December 5, 2018

SHARCNET™

About This Presentation

This presentation will discuss:

- how to create one or more pseudo-random number **generator** objects,
- how to create one or more **distribution** objects,
- how to properly **seed** a pseudo-random number generator,
- how to **use** such objects, and,
- show **examples** of using various distributions

that one may find useful when using ISO C++.

SHARCNET™

About This Presentation

- The commonly used C Standard Library defines:
  - `srand()` to “seed” the pseudo-random number generator, and,
  - `rand()` to “seed” generate the next pseudo-random number
- … but there are issues.

About This Presentation (con't)

c-style-rand1.cxx

```
1  #include <cstdlib>
2  #include <iostream>
3
4  int main()
5  {
6    using namespace std;
7
8    srand(5);                    // what seed should might one choose?
9    for (int i=0; i =/= 5; ++i)  // generate 5 numbers
10     cout << rand() << ' ';
11   cout << '\n';
12 }
```

About This Presentation (con't)

c-style-rand2.cxx

```
1  #include <cstdlib>
2  #include <iostream>
3
4  struct point { int x; int y; };
5
6  int main()
7  {
8    using namespace std;
9    srand(5);                      // what seed should might one choose?
10   for (int i=0; i != 5; ++i)     // generate random points...
11   {
12     point p{rand(),rand()};      // this breaks up the random sequence!
13     cout << '(' << p.x << ',' << p.y << ") ";
14   }
15 }
```

SHARCNET™

About This Presentation (con't)

c-style-rand3.cxx

```
1  #include <cstdlib>
2  #include <iostream>
3
4  int main()
5  {
6    using namespace std;
7    srand(5);                              // what seed should might one choose?
8    for (int i=0; i =/= 5; ++i)            // generate random points between 2 and 8...
9    {
10     int m1 = (rand() % (8-2)) + 2;       // is this a good way to do this?
11     int m2 = 8+rand()/(RAND_MAX/(8-2+1)+1); // or this?
12     // ...                               // or something else?
13   }
14 }
```

Using the C++ Standard Library

- ISO C++ defines its own header, `<random>`, for random-number generators and distributions.
- Each random number generator is an class:
  - capable of being initialized with appropriate seeds
  - whose state can be read from and written to a stream
  - permitting multiple independent random number sequences to be generated
- Each distribution properly maps a random number generator's output to that distribution.
- Provides `std::default_random_engine` if one simply wants a default random number engine.

Using the C++ Standard Library (con't)

demo1.cxx

```cpp
#include <random>
#include <iostream>

int main ()
{
  using namespace std;
  int const low = 1, high = 100, n = 10;

  random_device rd;                              // a device producing 32-bit random numbers
  uniform_int_distribution<int> dist(low,high);  // dist properly maps generator to [low,high]
  std::mt19937 generator(rd());                  // use rd() to choose a random 32-bit seed

  for (int i=0; i != n; ++i)                     // generate n random numbers...
    cout << ' ' << dist(generator);              // pumped through dist
  cout << endl;
}
```

SHARCNET™

Using the C++ Standard Library (con't)

demo2.cxx

```
1  #include <random>
2  #include <vector>
3  #include <iostream>
4  #include <algorithm>
5  using namespace std;
6  int main ()
7  {
8    int const low = 1, high = 100, n = 10;
9    random_device rd;                                  // a device producing 32-bit random numbers
10   vector<typename mt19937::result_type> data;        // Mersenne Twisters have >32-bits of seed state!
11
12   // populate a vector with random seed state...
13   generate_n(
14     back_inserter(data),                             // append new numbers into data
15     mt19937::state_size,                             // size of Mersenne Twister seed state
16     [&rd](){ return rd(); }                          // generate a single random seed value
17   );
```

SHARCNET™

Using the C++ Standard Library (con't)

```
18
19  // std::seed_seq is used to pass random seed state to mt19937...
20  seed_seq seed(begin(data), end(data));              // needed to pass to generator
21
22  std::mt19937 generator(seed);                       // choose a random 32-bit seed
23  uniform_int_distribution<int> dist(low,high);       // dist properly maps generator to [low,high]
24
25  for (int i=0; i != n; ++i)                          // generate n random numbers...
26    cout << ' ' << dist(generator);                   // pumped through dist
27  cout << endl;
28 }
```

Using the C++ Standard Library (con't)

random-utils.hxx

```cpp
#ifndef random_utils_hxx_
#define random_utils_hxx_
#include <random>
#include <vector>
#include <algorithm>

auto make_mersenne_twister() -> std::mt19937
{
  std::random_device rd;
  std::vector<typename std::mt19937::result_type> data;
  data.reserve(std::mt19937::state_size);
  generate_n(std::back_inserter(data), std::mt19937::state_size, [&rd](){ return rd(); });
  std::seed_seq seeds(begin(data), end(data));
  return std::mt19937(seeds);
}
#endif // #ifndef random_utils_hxx_
```

SHARCNET™

Using the C++ Standard Library (con't)

demo3.cxx

```
#include <random>
#include <iostream>
#include "random-utils.hxx"

int main()
{
  using namespace std;
  double const low = 1.1, high = 100.1;
  int const n = 10;
  auto rnd_num_gen = make_mersenne_twister();
  uniform_real_distribution<double> dist(low,high);
  for (int i=0; i != n; ++i)
    cout << ' ' << dist(rnd_num_gen);
  cout << endl;
}
```

SHARCNET™

Using the C++ Standard Library (con't)

Examples and discussion.