

Part of the Picture: Simulation

Random Number Generators—The RandomInt Class

The *Part of the Picture: Simulation* section in Chapter 5 referred to a class `RandomInt`, which can be used to conveniently generate random numbers. This source code for this class follows

Class `RandomInt` Documentation File.

```
/* RandomInt.doc documents RandomInt, a pseudo-random integer class.
 *
 * Written by: Joel C. Adams, Spring, 1993, at Calvin College.
 * Written for: C++: An Introduction To Computing.
 * Revised: Spring 1997, for C++: An Introduction To Computing 2e.
 *
 * Usage: to generate a...
 * nondeterministic sequence of 'random' numbers in 0..RAND_MAX:
 *     RandomInt rint1;
 * deterministic sequence of 'random' numbers in 0..RAND_MAX:
 *     RandomInt rint2(a);
 * nondeterministic sequence of 'random' numbers in b..c:
 *     RandomInt rint3(b, c);
 * deterministic sequence of 'random' numbers in b..c:
 *     RandomInt rint2(a, b, c);
 *
 * Caution: RandomInt objects are NOT autonomous, but SHARE the ANSI-C
 * standard random number generator rand(). For most applications,
 * this will have no effect. However, if a program defines an unseeded
 * RandomInt r1, and later defines a seeded RandomInt r2, then the
 * number-sequence for r1 will be 'random' prior to the definition
 * of r2, but deterministic following the definition of r2 (because r2's
 * re-seeding the random number generator affects the numbers of r1).
 *
 * Class Invariant: myRandomValue contains a 'random' number
 *                  between myLowerBound and myUpperBound (inclusive).
 *****/

// ***** RandomInt.h *****

#ifndef RANDOMINT
#define RANDOMINT

#include <iostream>           // ostream
#include <cstdlib>           // srand(), rand(), RAND_MAX
#include <cassert>           // assert()
#include <ctime>             // time()
using namespace std;

class RandomInt
{
public:                       // INTERFACE:
    RandomInt();              // constructors
    RandomInt(int low, int high); // default values
                                // bounded
};
```

```

RandomInt(int seedValue); // seeded
RandomInt(int seedValue, int low, int high); // seeded & bounded
// output:
void print(ostream & out) const; // member (<< below)
// generate next:
RandomInt generate(); // default bounds
RandomInt generate(int low, int high); // new bounds
// type-conversion
operator int(); // RandomInt-to-int

private: // PRIVATE SECTION:
// utility functions
void initialize(int low, int high); // unseeded initializer
void seededInitialize(int seedValue, // seeded initializer
int low, int high);
int nextRandomInt(); // formula encapsulation

int myLowerBound, // the minimum random value
myUpperBound, // the maximum random value
myRandomValue; // the current random value

static bool generatorInitialized; // initialization flag
};

/*****
* Initialize a RandomInt to a number within default bounds. *
* Postcondition: myLowerBound == 0 && myUpperBound == RAND_MAX && *
* srand() has been seeded (using time()) && *
* myRandomValue contains a 'random' number. *
*****/
inline RandomInt::RandomInt()
{
initialize(0, RAND_MAX);
}

/*****
* Initialize a RandomInt to a number within specified bounds. *
* Precondition: 0 <= low && low < high. *
* Postcondition: myLowerBound == low && myUpperBound == high && *
* srand() has been seeded (using time()) && *
* myRandomValue contains a 'random' number. *
*****/
inline RandomInt::RandomInt(int low, int high)
{
assert(0 <= low);
assert(low < high);
initialize(low, high);
}

/*****
* Initialize a RandomInt to a number within default bounds, *
* using a specified seed value. *
* Postcondition: myLowerBound == 0 && myUpperBound == RAND_MAX && *
* srand() has been seeded (using seedValue) && *
* myRandomValue contains a 'random' number. *
*****/
inline RandomInt::RandomInt(int seedValue)

```

```

{
    assert(seedValue >= 0);
    seededInitialize(seedValue, 0, RAND_MAX);
}

/*****
 * Initialize a RandomInt to a number within specified bounds
 * using a specified seed value.
 * Precondition: 0 <= low && low < high.
 * Postcondition: myLowerBound == low && myUpperBound == high &&
 *                srand() has been seeded (using seedValue) &&
 *                myRandomValue contains a 'random' number.
 *****/
inline RandomInt::RandomInt(int seedValue, int low, int high)
{
    assert(seedValue >= 0);
    assert(0 <= low);
    assert(low < high);
    seededInitialize(seedValue, low, high);
}

/*****
 * Output a RandomInt via an ostream (Function Member).
 * Precondition: out is an ostream.
 * Postcondition: myRandomValue has been inserted into out.
 *****/
inline void RandomInt::print(ostream & out) const
{
    out << myRandomValue;
}

/*****
 * Output a RandomInt via an ostream (Non-Function Member).
 * Precondition: out is an ostream,
 *               randInt is a RandomInt.
 * Postcondition: randInt.myRandomValue has been inserted into out &&
 *               ReturnValue == out.
 *****/
inline ostream & operator<< (ostream & out, const RandomInt & randInt)
{
    randInt.print(out);
    return out;
}

/*****
 * Generate next random integer.
 * Precondition: srand() has been called to seed rand() &&
 *               myLowerBound has been initialized &&
 *               myUpperBound has been initialized.
 * Postcondition: ReturnVal == a new 'random' number.
 *****/
inline int RandomInt::nextRandomInt()
{
    return myLowerBound + rand() % (myUpperBound - myLowerBound + 1);
}

/*****
 * Generate a new 'random' number within myLowerBound..myUpperBound.
 * Postcondition: myRandomValue has a new 'random' value &&
 *****/

```

```

*          ReturnValue == myself.          *
*****/
inline RandomInt RandomInt::generate()
{
    myRandomValue = nextRandomInt();
    return *this;
}

/*****
* Convert RandomInt objects to int objects, where appropriate.      *
* Precondition:  A RandomInt has been used in a context defined      *
*                for an int, but not for a RandomInt.                *
* Postcondition: myRandomValue has been read and returned.          *
*****/
inline RandomInt::operator int()
{
    return myRandomValue;
}

#endif

//***** RandomInt.cpp *****

bool RandomInt::generatorInitialized = false; // initialize the flag

/*****
* Initialize the data members and seed the random number generator    *
* using a default value (the clock).                                  *
* Precondition:  0 <= low && low < high.                              *
* Postcondition: myLowerBound == low && myUpperBound == high &&      *
*                srand() has been seeded (using time()) &&          *
*                MyRandomValue contains a 'random' number.          *
*****/
void RandomInt::initialize(int low, int high)
{
    myLowerBound = low;
    myUpperBound = high;

    if (!generatorInitialized) // call srand() first time only
    {
        srand(long(time(0))); // use clock for seed
        generatorInitialized = true;
    }

    myRandomValue = nextRandomInt();
}

/*****
* Initialize the data members and seed the random number generator    *
* using a specified value (seedValue).                                *
* Precondition:  0 <= low && low < high.                              *
* Postcondition: myLowerBound == low && myUpperBound == high &&      *
*                srand() has been seeded (using seedVal) &&        *
*                MyRandomValue contains a 'random' number.          *
*****/
void RandomInt::seededInitialize(int seedValue, int low, int high)

```

```

{
    myLowerBound = low;
    myUpperBound = high;
    srand(seedValue);
    generatorInitialized = true;
    myRandomValue = nextRandomInt();
}

/*****
 * Generate a new 'random' number within low..high.          *
 * Precondition:  0 <= low && low < high.                    *
 * Postcondition: myRandomValue has a new 'random' value &&   *
 *                ReturnValue == myself.                     *
 *****/
RandomInt RandomInt::generate(int low, int high)
{
    assert(0 <= low);
    assert(low < high);
    myLowerBound = low;
    myUpperBound = high;
    myRandomValue = nextRandomInt();
    return *this;
}

```

Class RandomInt Header File.

```

/* RandomInt.h declares RandomInt, pseudo-random integer class.
 *
 * Written by: Joel C. Adams, Spring, 1993, at Calvin College.
 * Written for: C++: An Introduction To Computing.
 * Revised: Spring 1997, for C++: An Introduction To Computing 2e.
 * Revised: Summer 2001, for C++: An Introduction To Computing 3e.
 *
 * See RandomInt.doc for class documentation,
 *     RandomInt.cpp for non-trivial operation definitions.
 *****/

#ifndef RANDOMINT
#define RANDOMINT

#include <iostream>
#include <cstdlib>
#include <cassert>
#include <ctime>
using namespace std;

class RandomInt
{
public:
    RandomInt();
    RandomInt(int low, int high);
    RandomInt(int seedValue);
    RandomInt(int seedValue, int low, int high);

    void print(ostream & out) const;

```

```

RandomInt generate();
RandomInt generate(int low, int high);
operator int();

private:
void initialize(int low, int high);
void seededInitialize(int seedValue, int low, int high);
int nextRandomInt();

int myLowerBound,
    myUpperBound,
    myRandomValue;

static bool generatorInitialized;
};

inline RandomInt::RandomInt()
{
    initialize(0, RAND_MAX);
}

inline RandomInt::RandomInt(int low, int high)
{
    assert(0 <= low);
    assert(low < high);
    initialize(low, high);
}

inline RandomInt::RandomInt(int seedValue)
{
    assert(seedValue >= 0);
    seededInitialize(seedValue, 0, RAND_MAX);
}

inline RandomInt::RandomInt(int seedValue, int low, int high)
{
    assert(seedValue >= 0);
    assert(0 <= low);
    assert(low < high);
    seededInitialize(seedValue, low, high);
}

inline void RandomInt::print(ostream & out) const
{
    out << myRandomValue;
}

inline ostream & operator<< (ostream & out, const RandomInt & randInt)
{
    randInt.print(out);
    return out;
}

inline int RandomInt::nextRandomInt()
{
    return myLowerBound + rand() % (myUpperBound - myLowerBound + 1);
}

```

```

inline RandomInt RandomInt::generate()
{
    myRandomValue = nextRandomInt();
    return *this;
}

inline RandomInt::operator int()
{
    return myRandomValue;
}

#endif

```

Class RandomInt Implementation File.

```

/* RandomInt.cpp defines the non-trivial RandomInt operations.

* Written by: Joel C. Adams, Spring, 1993, at Calvin College.
* Written for: C++: An Introduction To Computing.
* Revised: Spring 1997, for C++: An Introduction To Computing 2e.
* Revised: Summer 2001, for C++: An Introduction To Computing 3e.
*
* See RandomInt.h for the class declaration,
*   RandomInt.doc for class documentation.
*****/

#include "RandomInt.h"

bool RandomInt::generatorInitialized = false;

void RandomInt::initialize(int low, int high)
{
    myLowerBound = low;
    myUpperBound = high;

    if (!generatorInitialized)        // call srand() first time only
    {
        srand(long(time(0)));        // use clock for seed
        generatorInitialized = true;
    }

    myRandomValue = nextRandomInt();
}

void RandomInt::seededInitialize(int seedValue, int low, int high)
{
    myLowerBound = low;
    myUpperBound = high;
    srand(seedValue);
    generatorInitialized = true;
    myRandomValue = nextRandomInt();
}

```

```
RandomInt RandomInt::generate(int low, int high)
{
    assert(0 <= low);
    assert(low < high);
    myLowerBound = low;
    myUpperBound = high;
    myRandomValue = nextRandomInt();
    return *this;
}
```
