Generating Random Numbers in C++

There are many problems in which probability must be considered or statistical sampling must be used. For example, applications such as simple computer games and more involved gaming scenarios can only be described statistically. All of these statistical models require the generation of *random numbers*; that is, a series of numbers whose order cannot be predicted.

In practice, there are no truly random numbers. Dice never are perfect; cards are never shuffled completely randomly; the supposedly random motions of molecules are influenced by the environment; and digital computers can handle numbers only within a finite range and with limited precision. The best one can do is to generate *pseudorandom* numbers, which are sufficiently random for the task at hand.

All C and C++ compilers provide at least two library functions for creating random numbers; rand() and srand(). The rand() function produces a series of random numbers in the range 0 <= rand <= RAND_MAX, where the constant RAND_MAX is defined in the stdlib header file. The srand() function provices a starting "seed" value for rand(). If srand() or some other equivalent "seeding" technique is not used, rand() will always produce the same series of random numbers.

The general procedure for creating a series of N random numbers using C++'s library functions is illustrated by the following code:

```
srand(time(NULL));              // this generates the first "seed" value
for(int i = 0; i < N; i++)      // this generates N random numbers
   cout << rand() << endl;
```

The srand() and rand() functions each require the cstdlib header file. The time() function requires the time header file. In the code above, the argument to the srand() function is a call to the time() function with a NULL argument. With this argument the time() function reads the computer's internal clock time, in hundredths of seconds. The srand() function then uses this time, converted to an unsigned int, as a starting point for the generation of the pseudorandom numbers.

Scaling

One modification to the random numbers produced by the rand() function typically must be made in practice. In most applications either the random numbers are required as floating point values within the range 0.0 to 1.0 or as integers with a specified range, such as 1 to 100. The method for adjusting the random numbers produced by a random number generator to reside within such ranges is called *scaling*.

Scaling random numbers to reside within the range 0.0 to 1.0 is easily accomplished by dividing the returned value of rand() by RAND_MAX. Thus, the expression float(rand())/RAND_MAX produces a floating point number between 0.0 and 1.0.

Scaling a random number as an integer value between 0 and N is accomplished using the expression rand() % (N + 1). For example, rand() % 6 will produce a random number between 0 and 5.

Scaling a random number as an integer value between 1 and N may be accomplished by using the expression rand() % N + 1 or 1 + rand() % N. For example, 1 + rand() % 6 will produce a random number between 1 and 6.

A more general scaling expression would be L + rand() % N, where L is the lowest value you want. To generate random numbers between 55 and 60, you would use 55 + rand() % 6. An alternative, though lengthier formula, would be L + rand() % (H + 1 – L) where H represents the highest value and L represents the lowest value you want.

Here are some useful tips for using random numbers.

1. To get reasonably random numbers, remember to initialize the random number seed with srand() or an equivalent function using either an inputted value or the system time. Otherwise, you will always generate the exact same series of "random" numbers.

2. The srand() function call must appear before you generate any random numbers and must only appear in your program once. It should never be placed in a loop or in a function that could conceivably be called from a loop. So, put it in main() unless you have a really good reason to place it elsewhere.

3. The number you are dividing by in the modulus expression (the N) always represents the number of possible values you want to generate. Divide by 6 to generate random numbers between 0 and 5, 1 and 6, 50 and 55, etc.

Chuck Young

```
// RANDOM NUMBER GENERATER DEMO PROGRAM.
// Jim Engel
#include <iostream>
#include <iomanip>
#include <ctime>
using namespace std;
void main()
{
   int EightCount = 0;
   int randomNum;
   int kk, jj;
   srand(time(NULL));
   cout << "\n Random Numbers 1 to 2**16 :\n\n";

   for ( kk = 0; kk < 25; kk++ )         {
      for( jj = 0; jj < 20; jj++ )       {
         cout << setw(6) << rand();
      }
      cout << endl;
   }

   cout << "\n\nRandom Numbers 1 to 100:\n\n";

   for( kk = 0; kk < 25; kk++ )           {
      for(int jj = 0; jj < 40; jj++ )     {
         cout << setw(3) << rand()%101;
      }
      cout << endl;
   }

   cout << "\n\nSecond part\n\n";
   int count = 100000000;
   while ( count-- )                     {
      randomNum = rand();
      if( randomNum == 8 )
      ++EightCount;
   }
   cout << "Number of eights " << EightCount << endl;
}
```