Ekaba Bisong
Programming in C++
University of Calabar

Lesson Note #32
June 09, 2015

Adapted from C++ How To Program edited for our own purposes

Random Number Generation

The element of chance can be introduced into computer applications by using the C++ Standard Library function rand. Consider the following statement:

i = rand();

The function rand generates an unsigned integer between 0 and RAND_MAX (a symbolic constant defined in the <cstdlib> header). You can determine the value of RAND_MAX for your system simply by displaying the constant.

To produce integers in the range 0 to 5, we use the modulus operator (%) with rand as follows:

rand() % 6

This is called scaling. The number 6 is called the scaling factor. We then shift the range of numbers produced by adding 1 to our previous result.

```
#include <iostream>
#include <iomanip>
#include <cstdlib> // contains function prototype for rand 6 using namespace std;

int main()
{
   // loop 20 times
    for ( int counter = 1; counter <= 20; ++counter )
    {
        // if counter is divisible by 5, start a new line of output
        if ( counter % 5 == 0 )
            {
                cout << endl;
            } // end for
} //endmain
```

Output:

```
        6       6       5       5       6
        5       1       1       5       3
        6       6       2       4       2
        6       2       3       4       1
```

Function rand actually generates pseudorandom numbers. Repeatedly calling rand produces a sequence of numbers that appears to be random. However, the sequence repeats itself each time the program executes. Once a program has been thoroughly debugged, it can be conditioned to produce a different sequence of random numbers for each execution. This is called randomizing and is accomplished with the C++ Standard Library function ***srand***. Function ***srand*** takes an unsigned integer argument and seeds the rand function to produce a different sequence of random numbers for each execution. The new C++ standard provides additional random number capabilities that can produce nondeterministic random numbers—a set of random numbers that can't be predicted. Such random number generators are used in simulations and security scenarios where predictability is undesirable.

```
#include <iostream>
#include <iomanip>
#include <cstdlib> // contains function prototype for rand 6 using namespace std;
using namespace std;

int main()
{
    unsigned seed; // stores the seed entered by the user
    cout << "Enter seed: ";
    cin >> seed;
    srand( seed ); // seed random number generator

    // loop 10 times
    for ( int counter = 1; counter <= 10; ++counter )
    {
        // pick random number from 1 to 6 and output it
        cout << setw( 10 ) << ( 1 + rand() % 6 );

        // if counter is divisible by 5, start a new line of output
        if ( counter % 5 == 0 )
        {
            cout << endl;
        }
    }//endfor
}//endmain
```

```
Enter seed: 67
         6         1         4         6         2
         1         6         1         6         4
```

```
Enter seed: 432
         4         6         3         1         6
         3         1         5         4         2
```

```
Enter seed: 67
         6         1         4         6         2
         1         6         1         6         4
```

Let's run the program several times and observe the results. Notice that the program produces a different sequence of random numbers each time it executes, provided that the user enters a different seed. We used the same seed

in the first and third sample outputs, so the same series of 10 numbers is displayed in each of those outputs.

To randomize without having to enter a seed each time, we may use a statement like:

srand(time(0));

This causes the computer to read its clock to obtain the value for the seed. Function time (with the argument 0 as written in the preceding statement) typically returns the current time as the number of seconds since January 1, 1970, at midnight Greenwich Mean Time (GMT). This value is converted to an unsigned integer and used as the seed to the random number generator. The function prototype for time is in <ctime>.

We see that the width of the range is determined by the number used to scale rand with the modulus op- erator (i.e., 6), and the starting number of the range is equal to the number (i.e., 1) that is added to the expression rand % 6. We can generalize this result as:

number = shiftingValue + rand() % scalingFactor;

where ***shiftingValue*** is equal to the first number in the desired range of consecutive integers
and ***scalingFactor*** is equal to the width of the desired range of consecutive integers.