

same output. But it is possible to generate numbers that appear to be randomly generated; *i.e.*, numbers that are uniformly distributed within a given interval and for which there is no discernible pattern. Such numbers are called *pseudo-random numbers*.


The Standard C header file `<cstdlib>` defines the function `rand()` which generates pseudo-random integers in the range 0 to `RAND_MAX`, which is a constant that is also defined in `<cstdlib>`. Each time the `rand()` function is called, it generates another **unsigned** integer in this range.

EXAMPLE 4.26 Generating Pseudo-Random Numbers


This program uses the `rand()` function to generate pseudo-random numbers:

```
#include <cstdlib> // defines the rand() function and RAND_MAX const
#include <iostream>
using namespace std;

int main()
{ // prints pseudo-random numbers:
  for (int i = 0; i < 8; i++)
    cout << rand() << endl;
  cout << "RAND_MAX = " << RAND_MAX << endl;
}
```



```
1103527590
377401575
662824084
1147902781
2035015474
368800899
1508029952
486256185
RAND_MAX = 2147483647
```



```
1103527590
377401575
662824084
1147902781
2035015474
368800899
1508029952
486256185
RAND_MAX = 2147483647
```

On each run, the computer generates 8 **unsigned** integers that are uniformly distributed in the interval 0 to `RAND_MAX`, which is 2,147,483,647 on this computer. Unfortunately each run produces the same sequence of numbers. This is because they are generated from the same “seed.”

Each pseudo-random number is generated from the previously generated pseudo-random number by applying a special “number crunching” function that is defined internally. The first pseudo-random number is generated from an internally defined variable, called the *seed* for the sequence. By default, this seed is initialized by the computer to be the same value every time the program is run. To overcome this violation of pseudo-randomness, we can use the `srand()` function to select our own seed.

EXAMPLE 4.27 Setting the Seed Interactively

This program is the same as the one in Example 4.26 except that it allows the pseudo-random number generator's seed to be set interactively:

```
#include <cstdlib> // defines the rand() and srand() functions
#include <iostream>
using namespace std;

int main()
{ // prints pseudo-random numbers:
  unsigned seed;
  cout << "Enter seed: ";
  cin >> seed;
  srand(seed); // initializes the seed
  for (int i = 0; i < 8; i++)
    cout << rand() << endl;
}
```

Enter seed: 0

12345
1406932606
654583775
1449466924
229283573
1109335178
1051550459
1293799192

Enter seed: 1

1103527590
377401575
662824084
1147902781
2035015474
368800899
1508029952
486256185

Enter seed: **12345**

1406932606
654583775
1449466924
229283573
1109335178
1051550459
1293799192
794471793

The line `srand(seed)` assigns the value of the variable `seed` to the internal “seed” used by the `rand()` function to initialize the sequence of pseudo-random numbers that it generates. Different seeds produce different results.

Note that the seed value 12345 used in the third run of the program is the first number generated by `rand()` in the first run. Consequently the first through seventh numbers generated in the third run are the same as the second through eighth numbers generated in the first run. Also note that the sequence generated in the second run is the same as the one produced in Example 4.26. This suggests that, on this computer, the default seed value is 1.

The problem of having to enter a *seed* value interactively can be overcome by using the computer's system clock. The *system clock* keeps track of the current time in seconds. The `time()` function defined in the header file `<ctime>` returns the current time as an **unsigned** integer. This then can be used as the seed for the `rand()` function.

EXAMPLE 4.28 Setting the Seed from the System Clock

This program is the same as the one in Example 4.27 except that it sets the pseudo-random number generator's seed from the system clock.

Note: if your compiler does not recognize the `<ctime>` header, then use the pre-standard `<time.h>` header instead.

```
#include <cstdlib> // defines the rand() and srand() functions
#include <ctime>    // defines the time() function
#include <iostream>
//#include <time.h> // use this if <ctime> is not recognized
using namespace std;
int main()
{ // prints pseudo-random numbers:
  unsigned seed = time(NULL); // uses the system clock
  cout << "seed = " << seed << endl;
  srand(seed); // initializes the seed
  for (int i = 0; i < 8; i++)
    cout << rand() << endl;
}
```

Here are two runs using a UNIX workstation running a Motorola processor:

```
seed = 808148157
1877361330
352899587
1443923328
1857423289
200398846
1379699551
1622702508
715548277
```

```
seed = 808148160
892939769
1559273790
1468644255
952730860
1322627253
1305580362
844657339
440402904
```

On the first run, the `time()` function returns the integer 808,148,157 which is used to “seed” the random number generator. The second run is done 3 seconds later, so the `time()` function returns the integer 808,148,160 which generates a completely different sequence.

Here are two runs using a Windows PC running an Intel processor:

In many simulation programs, one needs to generate random integers that are uniformly distributed in a given range. The next example illustrates how to do that.

```
seed = 943364015
2948
15841
72
25506
30808
29709
13115
2527
```

```
seed = 943364119
17427
20464
13149
5702
12766
1424
16612
31746
```

EXAMPLE 4.29 Generating Pseudo-Random Numbers in Given Range

This program is the same as the one in Example 4.28 except that the pseudo-random numbers that it generates are restricted to given range:

```
#include <cstdlib>
#include <ctime>      // defines the time() function
#include <iostream>
//#include <time.h>   // use this if <ctime> is not recognized
using namespace std;

int main()
{ // prints pseudo-random numbers:
  unsigned seed = time(NULL);      // uses the system clock
  cout << "seed = " << seed << endl;
  srand(seed);                     // initializes the seed
  int min, max;
  cout << "Enter minimum and maximum: ";
  cin >> min >> max;               // lowest and highest numbers
  int range = max - min + 1;       // number of numbers in range
  for (int i = 0; i < 20; i++)
  { int r = rand()/100%range + min;
    cout << r << " ";
  }
  cout << endl;
}
```

Here are two runs:

```
seed = 808237677
Enter minimum and maximum: 1 100
85 57 1 10 5 73 81 43 46 42 17 44 48 9 3 74 41 4 30 68
```

```
seed = 808238101
Enter minimum and maximum: 22 66
63 29 56 22 53 57 39 56 43 36 62 30 41 57 26 61 59 26 28
```

The first run generates 20 integers uniformly distributed between 1 and 100. The second run generates 20 integers uniformly distributed between 22 and 66.