



Random Number Generation

RANDOM NUMBER GENERATION

- ❖ The element of chance can be introduced into computer applications by using the C Standard Library function `rand()` from the `<stdlib.h>` header.
- ❖ The `rand` function generates a pseudo-random integer between 0 and `RAND_MAX` (a symbolic constant defined in the `<stdlib.h>` header).
- ❖ It can be used as follows:

```
i = rand();
```

RANDOM NUMBER GENERATION

- ❖ Standard C states that the value of `RAND_MAX` must be at least 32767, which is the maximum value for a two-byte (i.e., 16-bit) integer.
- ❖ If `rand` truly produces integers at random, every number between 0 and `RAND_MAX` has an equal chance (or probability) of being chosen each time `rand` is called.
- ❖ The range of values produced directly by `rand` is often different from what is needed in a specific application.
 - For example, a program that simulates coin tossing might require only 0 for “heads” and 1 for “tails.”
 - A dice-rolling program that simulates a six-sided die would require random integers from 1 to 6.
 - `Dice.cpp`

RANDOM NUMBER GENERATION

- ❖ If we need our random numbers to be generated within a specific range, we can do that by using *scaling* and *shifting*.
- ❖ *Scaling* limits (i.e., “scales”) the maximum value produced by `rand()` to whatever we specify.
- ❖ *Shifting* shifts the range (generally to the right) on the number line.

Scaling

- ❖ To perform *scaling* on the value produced by the rand() function, we use the modulus operator:

```
int x = rand() % 6;
```

Produces a random number between 0 and 5.

- ❖ In general, a scaling factor of “n” produces random numbers between 0 and n-1.

```
int x = rand() % n;
```

Shifting

- ❖ To perform *shifting* on the value produced by the rand() function, we add an integer value to it:

```
int x = rand() + 5;
```

Produces a random number between 5 and RAND_MAX + 5.

Scaling and Shifting

- ❖ These two operations can be combined to produce a random integer value in almost any range we want.
- ❖ For example:

```
x = 1 + rand() % 6;
```

will produce random numbers between 1 and 6 (which might be useful if simulating a roll of the dice).

Scaling and Shifting

- ❖ **To summarize:** The values produced directly by `rand` are always in the range:
$$0 \leq \text{rand}() \leq \text{RAND_MAX}$$
- ❖ As you know, the following statement simulates rolling a six-sided die:
$$\text{face} = 1 + \text{rand}() \% 6;$$
 - The width of this range (i.e., the number of consecutive integers in the range) is 6 and the starting number in the range is 1.
- ❖ We can generalize this result as follows
$$n = a + \text{rand}() \% b;$$
- ❖ where a is the **shifting value** (which is equal to the first number in the desired range of consecutive integers) and b is the **scaling factor** (which is equal to the width of the desired range of consecutive integers).

Scaling and Shifting

- ❖ As we mentioned, the general formula for calculating random integers values within a specified range is:

$$n = a + \text{rand}() \% b;$$

where a is the **shifting value** and b is the **scaling factor**.

- ❖ Creating such a range is therefore a **two-step process**:
 - Determine the number of numbers in the desired range. Let that value = b .
 - ♦ Note that if $a \leq x \leq b$, the number of numbers = $(b - a) + 1$.
 - Determine the starting value of the range. Let that value = a .
- ❖ The resulting expression will generate random integer values over a range of size b and starting at a .

Scaling and Shifting

- ❖ Let's work some examples.... What C statements will produce random (integer) numbers in the following ranges?

- $13 \leq x \leq 25$

- $22 \leq x < 43$

- $17 < x \leq 29$

- $45 < x < 65$

$46 \leq x \leq 64$

$18 \leq x \leq 29$

$\text{rand}() \% 13 + 13$

≤ 42

$\text{rand}() \% 21 + 22$

$\text{rand}() \% 12 + 18$

$\text{rand}() \% 19 + 18$

Scaling and Shifting

❖ Let's work some examples.... What C statements will produce random (integer) numbers in the following ranges?

– $13 \leq x \leq 25$

♦ `13 + rand() % 13`

– $22 \leq x < 43$

♦ `22 + rand() % 21`

– $17 < x \leq 29$

♦ `18 + rand() % 12`

– $45 < x < 65$

♦ `46 + rand() % 19`



RANDOM NUMBER GENERATION

- ❖ Note that we get the exact same sequence of values each time we execute this program.
- ❖ How can these be random numbers? Ironically, this repeatability is an important characteristic of function `rand()`.
- ❖ When debugging a program, this repeatability is essential for proving that corrections to a program work properly.

RANDOM NUMBER GENERATION

- ❖ In truth, function `rand()` doesn't generate true random numbers. It actually generates what are called **pseudorandom numbers**.
 - These are numbers that appear to be random, but which actually repeat themselves each time the program is executed.
- ❖ As mentioned, this is useful for debugging purposes.
- ❖ But after a program has been thoroughly debugged, we may want it to produce a different sequence of pseudorandom numbers each time (rather than the same sequence each time).
- ❖ `randTestWithoutSeed.cpp`
- ❖ `randSeed.cpp`

RANDOM NUMBER GENERATION

- ❖ So, how do we make our programs produce a different sequence of random numbers each time?
- ❖ This is called **randomizing** and is accomplished with the standard library function `srand`.
- ❖ Function `srand` takes an **unsigned** integer argument and **seeds** function `rand` to produce a different sequence of random numbers for each execution of the program.
- ❖ `srand` is demonstrated in the following program.
 - Note that the function prototype for `srand` is found in `<stdlib.h>`.

Seed = 7
strand(7)

RANDOM NUMBER GENERATION

- ❖ To randomize without entering a seed each time, we can use a statement like:

```
srand( time( NULL ) );
```
- ❖ This causes the computer to read its clock to obtain the value for the seed automatically.
- ❖ Function `time()` returns the number of seconds that have passed since midnight on January 1, 1970.
- ❖ This value is converted to an unsigned integer and used as the seed to the random number generator.

RANDOM NUMBER GENERATION

- ❖ “`time_t`” is a system dependent data type representing time (generally an unsigned integer).
- ❖ The function prototype for `time()` is in `<ctime>`.

❖ `Rand_test1.cpp`

Add 2 Random numbers – Inclass activity

- ❖ Generate two random numbers. One number must be between 43 and 87. The second number must be between 35 and 93. Display the sum of two numbers