Random Number Generator and its Usage

Random numbers are very important in data encryption nowadays, and are also important in your project

Outline

1. **Bitwise operators in C**
2. Functions: rand(), srand(), etc.
3. Randomized algorithm
   - Monte Carlo simulation
   - Las Vegas Simulation
4. Algorithm:
   - Shuffling a deck of cards - Knuth shuffle

Bitwise operators in C: and, or, xor

&

```
int a = 5 ;
int b = 3 ;
c = a & b ;
```

```
    0101 (decimal 5)
AND 0011 (decimal 3)
  = 0001 (decimal 1)
```

```
    0011 (decimal 3)
AND 0010 (decimal 2)
  = [redacted] (decimal 2)
```

|

```
int a = 5 ;
int b = 3 ;
c = a | b ;
```

```
   0101 (decimal 5)
OR 0011 (decimal 3)
 = 0111 (decimal 7)
```

```
   0010 (decimal 2)
OR 1000 (decimal 8)
 = [redacted] (decimal 10)
```

^

```
int a = 2 ;
int b = 10 ;
c = a ^ b ;
```

```
    0010 (decimal 2)
XOR 1010 (decimal 10)
  = 1000 (decimal 8)
```

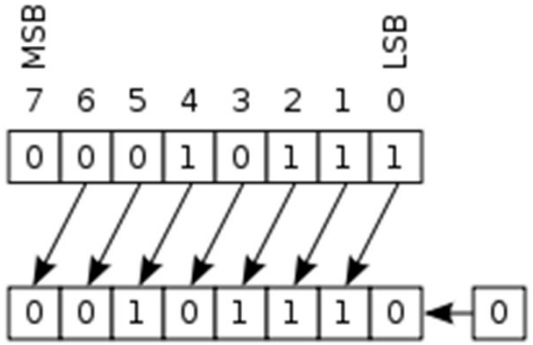
```
    0101 (decimal 5)
XOR 0011 (decimal 3)
  = [redacted] (decimal 6)
```

Note: ^ in C does not mean "power".
It means XOR (exclusive OR) – **returns true only if one of the two operands is true**.

Bitwise operators in C: shift

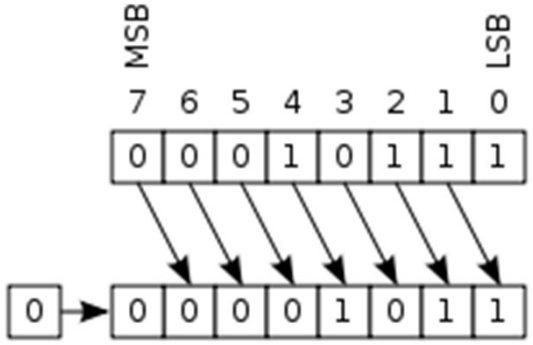
<<

```
    00010111 LEFT-SHIFT
 =  00101110
```



>>

```
    00010111 RIGHT-SHIFT
 =  00001011
```



* Good also for multiplication and division by two

Bitwise operators in C

operator	meaning	examples
&	bitwise AND	`3 & 2 gives` ??
|	bitwise OR	`2 | 1 gives` ??
^	bitwise XOR	`3 ^ 2 gives` ??
~	NOT	`~3 gives` ??
<<	shift left	`3 << 1 gives` ??
>>	shift right	`3 >> 1 gives` ??

- "NOT" is something called 2's complement; we saw it in Lecture 2b
- "XOR" has an interesting property:
  - Given integers D (data) and K (key), and E = D ^ K (encrypted)
  - What is E ^ K?

E.g., 1100 ^ 1010 = 0110, then 0110 ^ 1010 = 1100

Let D=1100 and K=1010

(D ^ K) ^ K = D

Note: https://en.wikipedia.org/wiki/Two%27s_complement

Outline

1. Bitwise operators in C
2. **Functions: rand(), srand(), etc.**
3. Randomized algorithm
   - Monte Carlo simulation
   - Las Vegas Simulation
4. Algorithm:
   - Shuffling a deck of cards - Knuth shuffle

Using `rand()`

- The **`rand()`** function generates a number randomly between **`0`** **`&`** **`RAND_MAX`**, inclusively.

RAND_MAX is defined in **stdlib.h** as 2,147,483,647 in Mac OS X , but as 32,767 or 2,147,483,647 in different Visual Studio versions.

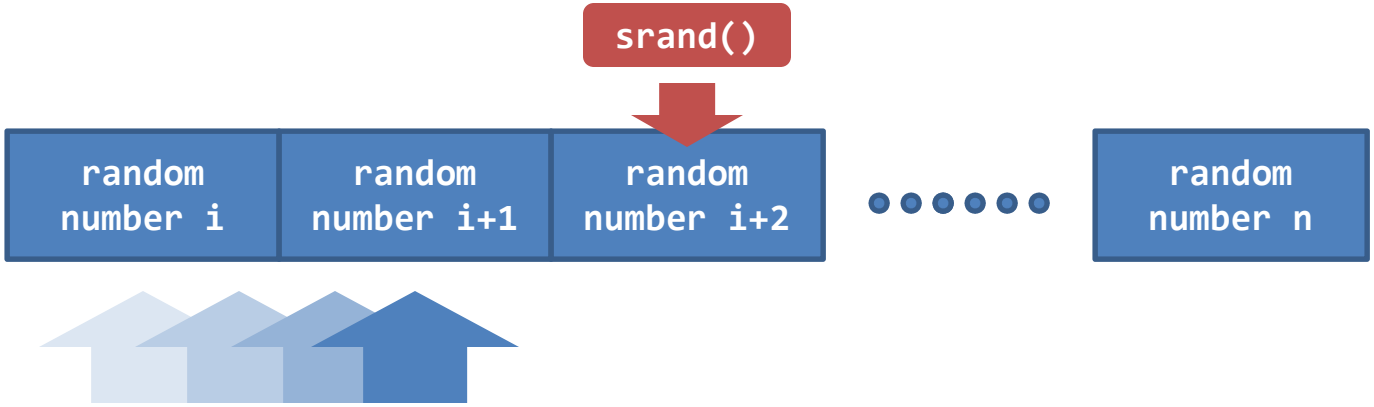
OMG! Where is the randomness?

rand_1.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main( void )
5  {
6      printf( "%d\n" , rand() );
7      return 0 ;
8  }
```

```
[1-st run]
16807
[2-nd run]
16807
[3-rd run]
16807
```

Concept: Pseudo-random number generator



- The truth about the random numbers: there is a sequence of predictable numbers!!!
  - ***rand()*** *always starts at the same position in the list,* and uses the current number to produce the next one!
  - **srand()** is to re-position the starting point (seed)

A must-watch movie https://www.youtube.com/watch?v=SxP30euw3-0

Pseudo-random number generator

- One common strategy: set the new position (seed) as the current time by the **`time()`** function.

rand_2.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main( void )
{
    srand( time( NULL ) );
    printf( "%d\n" , rand() );
    return 0 ;
}
```

```
[1-st run]
135790
[2-nd run]
246801
```

`time(NULL)` returns the current time, in terms of the number of seconds since 00:00, 1970 Jan 1 GMT, or the **Epoch Time**.

Note:

- If you input the same seed to **`srand()`**, you'll always get the same random sequence!
- However, this could be good for debugging, i.e., you keep the same random value.

An example PRNG

From https://en.wikipedia.org/wiki/Xorshift

```c
#include <stdio.h>
#include <stdint.h>

uint64_t seed = 1002 ;   // seed must start with a nonzero value
uint64_t xorshift64star( void )
{
    // >>, <<, and ^ are bitwise operator
    seed ^= seed >> 12 ;        // >> - shift to right
    seed ^= seed << 25 ;        // << - shift to left
    seed ^= seed >> 27 ;        // ^  - XOR (exclusive OR)
    seed  = seed * UINT64_C( 2685821657736338717 ) ;
    return seed ;
}
int main( void )
{
    printf( "%d\n" , (int) ( xorshift64star() % 1000 ) );
    printf( "%d\n" , (int) ( xorshift64star() % 1000 ) );
    printf( "%d\n" , (int) ( xorshift64star() % 1000 ) );
    return 0 ;
}
```

Using PRNG

- **Question 1 (or a fair coin)**: Generate a random integer of either 0 or 1.

```
int number = rand() % 2 ;
```

- **Question 2**: Generate a random integer distributed uniformly in the range [1, 6]?

```
int number = rand() % 6 + 1 ;
```

Using PRNG

- **Question 3**: Generate a random integer distributed uniformly in the range **[X, Y]**, where **X** & **Y** are user inputs and **X < Y**?

```
int number = rand() % (Y-X+1) + X ;
```

Using PRNG

- **Question 4**: Generate a random floating-point numbers distributed uniformly in the range [0, 1]?

```
double number = rand() / (double) RAND_MAX ;
```

- **Question 5**: Generate a random floating-point numbers distributed uniformly in the range [0, 1)?

```
double number = rand() / ( (double) RAND_MAX + 1 );
```

Outline

1. Bitwise operators in C
2. Functions: rand(), srand(), etc.
3. **Randomized algorithm**
   - **Monte Carlo simulation**
   - **Las Vegas Simulation**
4. Algorithm:
   - Shuffling a deck of cards - Knuth shuffle

Randomized algorithm

Doing computation in a random way!!!

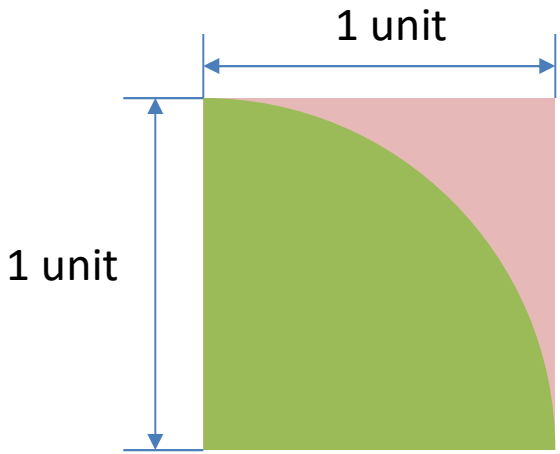
- Class A: **Monte Carlo** Simulation
  - A randomize algorithm that does not guarantee correctness; it guarantees **a bounded running time**.
- Class B: **Las Vegas** Simulation
  - A randomize algorithm that does not guarantee its running time; it guarantees **a bounded correctness**.
  - Always gives correct results but gambles with resources.

Examples

- Example: Monte Carlo Simulation
  - Among N students, we want to find the tallest one
  - Algorithm: we randomly pick a student and update the max height seen so far; the more we try, more correct
  - Maybe incorrect with a certain probability

- Example: Las Vegas Simulation
  - Among N students, we want to find Peter (only one)
  - Algorithm: we randomly pick a student and ask his/her name until we find Peter; but gamble with time taken.

Uniform Sampling – Find PI

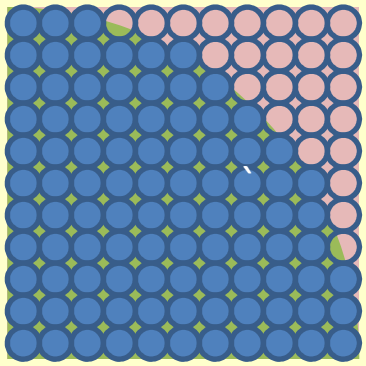


Calculation of π

$$\frac{\text{Area of Sector}}{\text{Area of Square}} = \frac{\pi \times r^2}{4} \Big/ 1$$

$$\pi = \frac{\text{Area of Sector}}{\text{Area of Square}} \times 4$$

$$= \frac{\text{Dots in green area}}{\text{Total number of dots}} \times 4$$

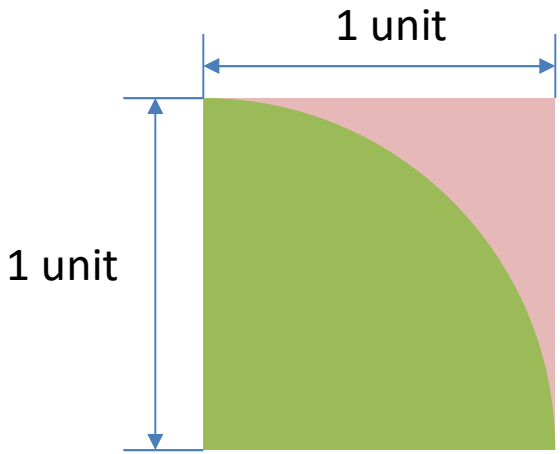


Total: 11 x 11 = 121 dots

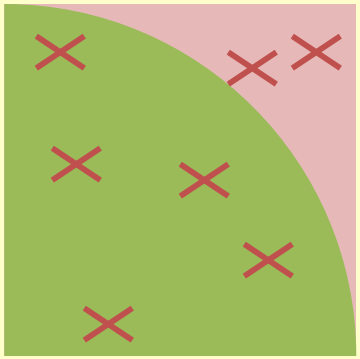
Number of dots inside: 95

So, π = 95 / 121 x 4= 3.140496

Which Type??? – Find PI



Dart throwing!!!



Calculation of π

$$\frac{\text{Area of Sector}}{\text{Area of Square}} = \frac{\pi \times r^2}{4} \Big/ 1$$

$$\pi = \frac{\text{Area of Sector}}{\text{Area of Square}} \times 4$$

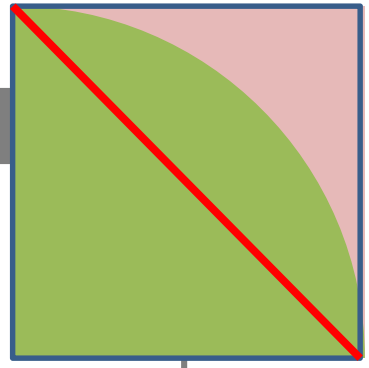
$$= \frac{\text{Dots in green area}}{\text{Total number of dots}} \times 4$$

- Randomize a set of points (as above)
- Count the points (darts) that are inside the green area
- Use the previous calculation to compute the estimated π value.

The more the points are, the more accurate the result becomes!

Monte Carlo simulation – How?

- We call the following pseudo-code.



Find PI

```
1  // Target Generate 100,000,000 darts
2
3  SET in_circle_count to 0
4
5  FOR i = 0 to 100,000,000 – 1 DO
6
7      SET x_coord to Randomly generate a number from 0 to 1
8      SET y_coord to Randomly generate a number from 0 to 1
9
10     IF ( x_coord , y_coord ) is inside the circle THEN
11         SET in_circle_count to in_circle_count + 1
12     END IF
13
14 END FOR
15
16 SET Result = in_circle_count / 100,000,000 * 4
```

Any idea to speed up the computation?

Hint: see red line above!

Outline

1. Bitwise operators in C
2. Functions: rand(), srand(), etc.
3. Randomized algorithm
   - Monte Carlo simulation
   - Las Vegas Simulation
4. **Algorithm:**
   - **Shuffling a deck of cards - Knuth shuffle**

Shuffling a deck of cards?

- This is a very common interview question from companies like Google, Facebook, Microsoft, etc.
  - Given a deck of **N** cards, where **N > 1**
  - Describe a set of procedure (or write a program) to shuffle them randomly

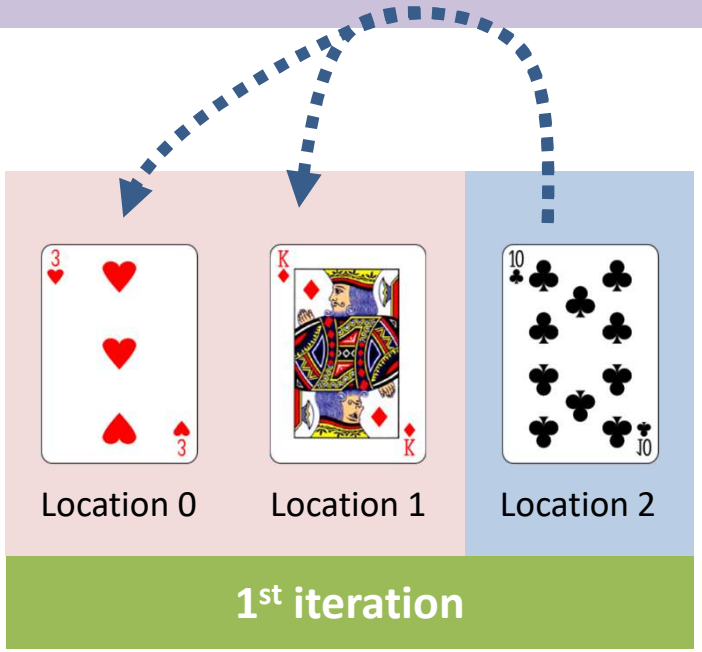
Shuffling a deck of cards?

- What is the definition of "***shuffle randomly***"?
  - Given a deck: { 1 , 2 , 3 }
  - What are the possible permutations?

1 2 3	1 3 2	2 1 3
2 3 1	3 1 2	3 2 1

  - By running your shuffling algorithm repeatedly, **your algorithm should produce approximately the same number of occurrences** for each of the above permutations.

Knuth shuffle – How it works



In the first iteration:

Random number j = { 0 , 1 , 2 }

To swap with the 1st/2nd card (if j = 0, 1)

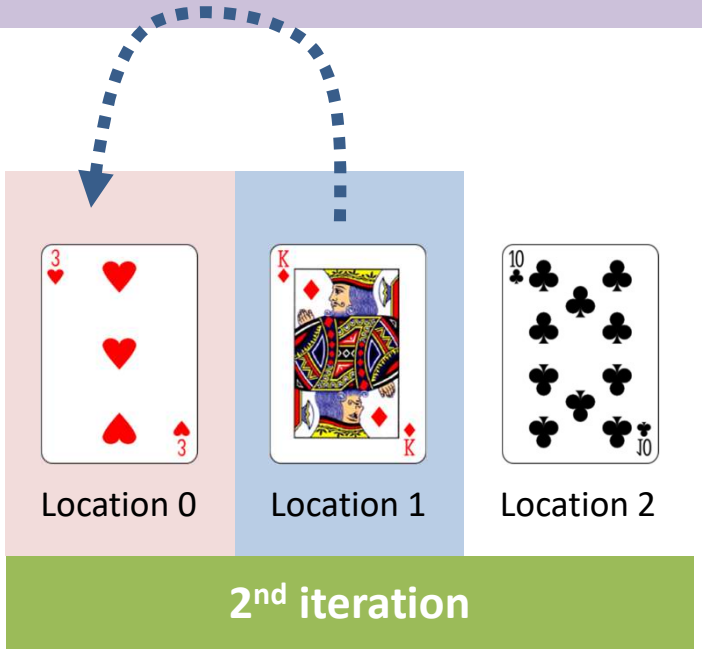
or

Not to swap (if j = 2)

Knuth shuffle of N cards

```
1  FOR i = n – 1 to 1 DO
2      Choose j randomly with 0 <= j <= i
3      Swap Card i and Card j
4  END FOR
```

Knuth shuffle – How it works



In the second iteration:

Random number j = { 0 , 1 }

To swap with the 1st card (if j = 0)

or

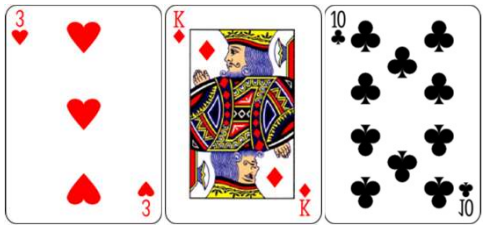
Not to swap (if j = 1)

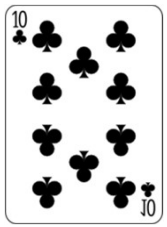
Knuth shuffle of N cards

```
1  FOR i = n – 1 to 1 DO
2      Choose j randomly with 0 <= j <= i
3      Swap Card i and Card j
4  END FOR
```

Knuth shuffle – Why it works

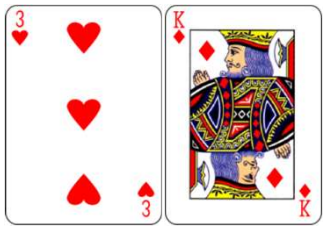
- 1st iteration: select **1 out of N cards** randomly

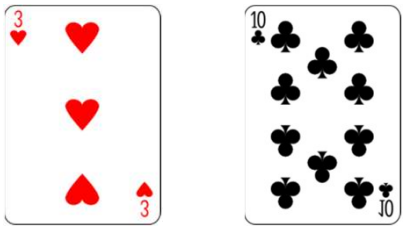




Set of fixed cards

- 2nd iteration: select **1 out of N-1 cards** randomly

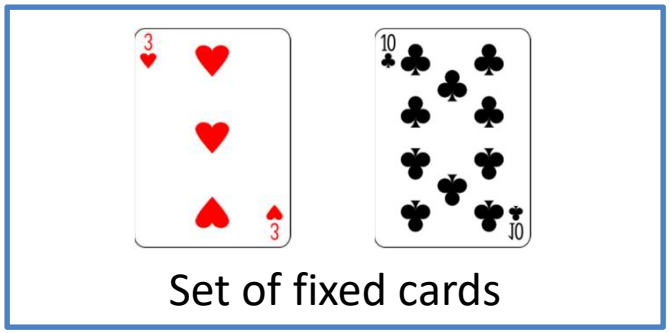




Set of fixed cards

Knuth shuffle – Why it works

- Therefore, the deck of N cards:
  - In the i-th iteration, we choose **1 out of (N-i+1) cards** randomly.
  - Note importantly that: **a card in the set of fixed cards should not be chosen again**!



Set of fixed cards

Knuth shuffle – Why it works

- Then, using Knuth Shuffle, the probability of generating a particular permutation is:

`1 / N!`

- If the random number generator produces a fairly uniform distribution:

All "N!" permutation will have an equal chance of being generated.

Knuth shuffle – Write it!

- **Challenge #1.** Can you write a program to show that Knuth Shuffle **generates all N! permutation with (approx.) equal chances**?

- **Challenge #2.**

  Can you find the bug in the following program?

```
1  int i , j , data[ 5 ] = { 1 , 2 , 3 , 4 , 5 };
2  for ( i = 4 ; i > 0 ; i-- )
3  {
4      j = rand() % 5 ;       // always generate [0,4]
5      SWAP( data , i , j ); // a func. to swap data[ i ] & data[ j ]
6  }
```

Reference: http://en.wikipedia.org/wiki/Fisher%E2%80%93Yates_shuffle#Implementation_errors

Knuth shuffle – Write it!

- **Challenge #1.** Can you write a program to show that Knuth Shuffle **generates all N! permutation with (approx.) equal chances**?

- **Challenge #2.**

  Bug fixed!

```
1  int i , j , data[ 5 ] = { 1 , 2 , 3 , 4 , 5 };
2  for ( i = 4 ; i > 0 ; i-- )
3  {
4      j = rand() % ( i+1 );
5      SWAP( data , i , j ); // a func. to swap data[ i ] & data[ j ]
6  }
```

Reference: http://en.wikipedia.org/wiki/Fisher%E2%80%93Yates_shuffle#Implementation_errors

Final note…

- Pseudo Random vs True Random
- If you want “high-quality” random numbers, e.g., for network security and cryptography, never just call rand()……

See more:

- Samsung 5G phone in 2020: QRNG (Quantum Random Number Generator) chip: https://www.sammobile.com/news/skt-galaxy-a71-5g-quantum-rng-chip-advanced-security/
- https://www.youtube.com/watch?v=SxP30euw3-0
- http://spectrum.ieee.org/semiconductors/processors/behind-intels-new-randomnumber-generator
- http://www.random.org/