Chapter 9

Random Numbers

Can a deterministic machine such as a computer generate truly "random" numbers?
In fact what generally passes for random numbers are sequences of independent random numbers, with a specified distribution.
Such sequences generated deterministically are often called "psuedorandom" or "quasirandom".
A good source on random numbers is:
Donald Knuth: "The art of Computer Programming" Volume 2

9.1 Random Number Generators

Random numbers are frequently required

- initialising a system before simulation
- data encryption
- choosing the order in which experimental data is analysed, avoiding bias
- rolling dice, shuffling cards
- "Monte Carlo method" QM and QFT

The C system provides a random number generator
It is a function called `rand()`
generates random integers between [0, `RAND_MAX`], where `RAND_MAX` is system dependent and defined in `stdlib.h`.
Therefore we need `#include<stdlib.h>` in programs using rand().
See the value of `RAND_MAX` by writing a program with

```
printf("%d\n",RAND_MAX)
```

you should get 2147483647 (ie $2^{31} - 1$)
In programming applications you might need to fill an array/matrix with random numbers eg.

```
#include<stdio.h>
#include<stdlib.h>
#define N 10

main()
{
   int ran_nums[N], i;

   for(i=0;i<N;i++)
         {
```

```
            ran_nums[i]=rand();
            printf("%d\n",ran_nums[i]);

        }

}
```

This prints out 10 random numbers. Run this 100 times and you will still get the exact same list of 10 numbers.
For most applications this isn't good say we want 2 arrays of random numbers bu don't want two identical arrays.
To understand why this happens we dissect the function.

rand() is a Linear congruential generator. This means that it generates a sequence of ints I_1, I_2, I_3, ... each between [0, `RAND_MAX`] using the recurrence relation

$$I_{j+1} = (aI_j + c) \bmod m$$

for

m	modulus:	$0 < m$:	2^{32}
a	multiplier:	$0 \leq a < m$:	1103515245
c	increment:	$0 \leq c < m$:	12345

The "seed" is the initial I_0 that starts the sequence.
Change the seed change the sequence.
m,a,c are chosen to maximise the sequence length, ie a very large number of "random numbers" would have to be generated before repetition is observed.
to see the generator.

1. choose a value for the seed
2. intialise the generator with this seed

Eg

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



```
#include<stdlib.h>
#include<time.h>
#define N 10

main()
{
   int a[N][N], i,j,seed;
   seed=time(NULL);
   srand(seed);
   for(i=0;i<N;i++)
         {
           for(j=0;j<N;j++)
           {
             a[i][j]=rand();
             printf("%d\n",a[i][j]);

         }
        }

}
```

Notes
`time.h` $\Rightarrow$ a C library file
contains functions dealing with
date, time, internal clock,...
the function call
`time(NULL)`
returns the number of seconds since 1 Jan 1970.
Repeated calls to `rand()` generates all ints in [0,`RAND_MAX`] in a mixed up order.

The value used as seed determines where in the sequence `rand()` starts off.
Using '`time(NULL)`' as seed therefore every time the program runs the seed

is different which means a different list of numbers.

9.1.1 Quick recap

rand()	generates a random number
srand()	seeds the generator with an integer value given by 'seed'
time(NULL)	returns the number of seconds since 01 Jan 1970

9.2 Non integer random numbers

We have seen that `rand()` generates random integer values.
How can we adapt this to get a real number between 0.0 and 1.0
We could

```
float x;


x= rand()/(RAND_MAX+1);
```

Now, say we want to generate lotto numbers?
These occur in a range 1-42 so we need to restrict the output of `rand()` to ints in [1,42] from the range `[0,RAND_MAX]`
Therefore we use the modulus function "%"
Recall

$$5\%3 = 2$$

ie the remainder when 5 is divided by 3
Therefore we use the function

```
([rand()%42]+1)
```


which should generate nos in [1,42].

we need '+1' if	`rand()% 42=0`
	then `rand()%42+1=1`
and if	`rand()%42=41`
	then `rand()%42+1=1`
and	
	% is above + in preference table
	⇒ % operation done first

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#define N 6

main()
{
   int lotto[N], i;
   seed=time(NULL);
   srand(seed);
   for(i=0;i<N;i++)
         {
             lotto[i]=rand()%42 +1;
             printf("%d\n",lotto[i]);

         }
}
```

Note: an alternative to `rand()%42 +1` is

```
lotto[i]=1+ (int)(42.0+rand()/(RAND_MAX+1.0));
```

There are many other random number generators on the market. See Numerical Recipes section 7 for discussion.

One simple variant is the "Multiplication Congruential" algorithm:

$$I_{j+1} = aI_j \pmod{\text{m}}$$

a and m must be chosen carefully to maximise the length of the sequence I_0, I_1, I_2,...
Recommended:

$$a = 7^5 = 16807$$

$$m = 2^{31} - 1$$

Note that aI_j may produce a number requiring more than 32-bits.
So either write rand() using 64-bit in the function
or
use clever tricks to factorise large numbers to sums of smaller ones.

Mersenne Twister

The Mersenne twister is a pseudorandom number generator developed in 1997 by Makoto Matsumoto and Takuji Nishimura

It provides for fast generation of very high quality pseudorandom numbers, having been designed specifically to rectify many of the flaws found in older algorithms.

There are at least two common variants of the algorithm, differing only in the size of the Mersenne primes used. The newer and more commonly used one is the Mersenne Twister MT 19937.

Mersenne Primes A Mersenne prime is a prime number that is one less than a prime power of two. For example, 31 (a prime number) = 32-1 = 2^5 - 1, and 5 also a prime number, so 31 is a Mersenne prime; so is 7 = 8 - 1 = 2^3 - 1. On the other hand, 2047 = 2048 - 1 = 2^{11} - 1, for example, is not a prime, because although 11 is a prime (making it a candidate for being a Mersenne prime), 2047 is not prime (it is divisible by 89 & 23). Throughout modern times, the largest known prime number has very often been a Mersenne prime.

More generally, Mersenne numbers (not necessarily primes, but candidates for primes) are numbers that are one less than a prime power of two; hence,

$$M_n = 2^n - 1.$$

(most sources restrict the term Mersenne number to where n is prime as all Mersenne primes must be of this form as seen below). It is currently unknown whether there is an infinite number of Mersenne primes.

See `http;\\www.mersenne.org`.

MT 19937 has the following desirable properties:

- It was designed to have a colossal period of $2^{19937} - 1$ (the creators of the algorithm proved this property). This period explains the origin of the name: it is a Mersenne prime, and some of the guarantees of

the algorithm depend on internal use of Mersenne primes. In practice, there is little reason to use larger ones.

- It has a very high order of dimensional equidistribution (compared to linear congruential generator).

  A bounded sequence $s_n, n = 1, 2, 3, \ldots$ of real numbers is equidistributed on an interval (a, b) precisely if for any subinterval (c, d) we have

  $$\lim_{n\to\infty} \frac{|s_1, s_2, \ldots, s_n \cap (c, d)|}{n} = \frac{d-c}{b-a}$$

  i.e. the proportion of terms falling in any subinterval is proportional to the length of the subinterval. Note that this means, by default, that there is negligible serial correlation between successive values in the output sequence.

- It is faster than all but the most statistically unsound generators.

- It is statistically random in all the bits of its output, and passes the stringent "Diehard tests".