C PROGRAMMING: RANDOM NUMBER GENERATION

We're about to start the core of our cluster: random walks. We first need to learn how to generate random numbers. Some people object to calling numbers random when they are generated with a computer code which has definite steps. These people prefer the name pseudo-random numbers. I'm not sure it's such a big deal, really. After all, you could make a similar objection to calling a coin toss or die roll a random process, since, really, the motion of a spinning coin or die is completely determined by Newton's laws of motion and so the outcome is not really random either.

We will look at two random number generators. The first is not as good a generator. (There are various tests of how 'random' the generator really is, and the first one scores less well on these tests.) It has the virtue however that you can see exactly what it is doing. The second is better, and it's the one we will therefore use. However, it is a 'black box': we just call some code someone else wrote and we cannot peak inside. So it's good to start with the more open one.

The first code, on page two, generates (pseudo) random numbers this way: you enter a seed integer i to start the process. Then you multiply i by 7^5 and calculate the remainder when it is divided by $2^{31}-1$. Finally, you divide the result by $2^{31}-1$. (In the last step, make sure you do this as real number arithmetic and not as integer arithmetic. This is why the code has separate variables M and rM.) That is

$$\begin{aligned} i &= 7^5 * i \\ i &= i \,\%\, (2^{31}-1) \\ i &= i/(2^{31}-1) \end{aligned}$$

To get another random number you repeat the process using your new i as the seed for the next step. You can continue this process as long as you like. Well, not really: Eventually the (pseudo) random numbers start to repeat. This is one reason this generator is not such a good one. In fact, can you see the maximum number of different number of randoms that can be generated with this algorithm? Hint: Think about how many possible remainders you can get when you divide by something.

```
/* linear congruential random number generator */
#include <stdio.h>
#include <math.h>
int main(void)
{
double r,rM;
long unsigned int a,b,i,M,j,N;
printf("\nEnter seed");
printf("\n");
scanf("%i",&i);
printf("\nEnter number of iterations");
printf("\n");
scanf("%i",&N);
a=7*7*7*7*7;
M=  2*2*2*2*2*2*2*2;
M=M*2*2*2*2*2*2*2*2;
M=M*2*2*2*2*2*2*2*2;
M=M*2*2*2*2*2*2*2*2  -1;
rM=M;
for (j=0;j<N;j=j+1){
i=a*i;
i=i%M;
r=i;
r=r/rM;
printf("\n%12.8lf",r);
}
printf("\n");
return 0;
}
```

Comments:

[1] Type in the code and generate 20 random numbers to see what they look like.

```
/* This program generates N random numbers between 0 and 1 */
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
int main(){
srand(time(NULL));
        int i,N;
double R;
printf("Enter the number of random numbers you want ");
scanf("%d",&N);
for(i=0;i<N;i++)
{
R=(double)rand()/RAND_MAX;
        printf(" %lf\n",R);
}
return 0;
}
```

[2] This random numer generator uses the computer's clock to provide a seed. Hence the need for the extra header file 'time.h'.

[3] Type in the code and generate 20 random numbers to see what they look like.

[4] Our random numbers are uniform on $[0, 1]$. That is they obey $0 < r < 1$ and the r are equally likely to be anywhere in the interval. There are other random number generators one can construct, but this is the most common type.