Pseudorandom Number

計算機只能接受明確指令，人類只好使用明確數學公式、明確演算法來製造隨機數。隨機數具有規律、可以預測，於是早期文獻稱做「偽隨機數」、「偽亂數」。

儘管隨機數具有規律、可以預測，我們還是可以努力讓隨機數「看起來似乎很亂」，讓人一時無法預測。但由於缺乏「亂」的明確定義，所以無法精準評比優劣。一切都是靠感覺，很不科學。

C 的內建函式庫，提供了隨機數。可惜不是均勻分布 uniform distribution ，時常為人詬病。

#include <time.h>

void random\_number()

{ // 使用特定編號的隨機數產生公式。// srand(18);

srand(time(0)); // 以現在時刻決定編號，盡量讓每次執行結果不同。

// 印出100個隨機數，範圍是[0, RAND\_MAX)。不是均勻分布。

for (int i=0; i<100; ++i) cout << rand();

// 印出100個隨機數，範圍是[0, 10)。完全不是均勻分布。

for (int i=0; i<100; ++i) cout << rand() % 10;

}

C++ 的內建函式庫，提供了健全的隨機數。

#include <random>

void random\_number()

{ // 使用特定的隨機數產生公式。

default\_random\_engine g;

// 使用平均分布，範圍[0, 10)，整數。

uniform\_int\_distribution<int> rand(0, 9);

// 印出100個整數隨機數，範圍是[0, 10)。

for (int i=0; i<100; ++i) cout << rand(g);

// 使用平均分布，範圍[0, 1]，浮點數。

uniform\_real\_distribution<float> uniform(0, 1);

// 印出100個浮點數隨機數，範圍是[0, 1]。

for (int i=0; i<100; ++i) cout << uniform(g);

}

演算法（ Linear Congruential Generator ）

先備知識是「餘數」。

以數學式子 xnext = (((x ⋅ a) + b) % n); 不斷製造隨機數。

當 n 是質數，則 0 到 n-1 恰好各出現一次，呈均勻分布。不過當 n 不是質數，就很難說了，必須小心設定 a b n 才行。

缺點：每 n 個數字循環出現，可以預測。實務上會讓 n 很大，以避開缺點。

C 語言的 rand() 即是此演算法。

// 得到[0, 2³²-1]之間的隨機數

unsigned int random()

{ // a = 196314165, b = 907633515, n = 2³²

static unsigned int seed = 0;

return seed = (seed \* 196314165) + 907633515;

}

// 得到[0,1]之間的隨機數

float random()

{ static unsigned int seed = 0;

seed = (seed \* 196314165) + 907633515;

// 請參考IEEE 754浮點數規範。

// 讓前九個最高位元是001111111，其餘是隨機數，

// 如此可以產生1到2之間的隨機數。

static unsigned int number = 0;

number = (seed >> 9) | 0x3F800000;

return (\*(float\*)&number) - 1.0f;

}

// 得到[-1,+1]之間的隨機數

float white\_noise()

{

static unsigned int seed = 0;

seed = (seed \* 196314165) + 907633515;

// 讓前九個最高位元是010000000，其餘是隨機數，

// 如此可以產生2到4之間的隨機數。

static unsigned int number = 0;

number = (seed >> 9) | 0x40000000;

return (\*(float\*)&number) - 3.0f;

}

Write a program to simulate throwing darts. (射鏢遊戲)

1

1

0

Use a random number generator to obtain 1,000 pairs of floating-point numbers (*x, y*) satisfying 0< *x<*1, 0*< y<*1.

Print the proportion *P* of throws that hit the dart board, that is, the proportion of pairs (*x, y*) that are inside the circle. Also print 4\* *P.*

Notice that the geometry of the problem leads us to expect *P* to be about . Thus 4\* *P* provides an approximation of .

Note: You can use the following process to generate random number

between 0~1:

double seed;

const double mpy = 25173.0;

const double inc =13849.0;

const double mod =65535.0;

input variable “seed” then calculate the following formula:

**seed = (seed \*mpy + inc) % mod ; // fmode(seed \*mpy + inc, mod)**

then **get one random number** between 0~1 by using: **seed/mod**