這篇文章主要引導初學者使用亂數，同時附上常被翻出來討論的議題，C/C++適用，唯以 C 語言撰之。

也由於是引導初學者，所以在某些用詞上會較不正確，

像 compiler、IDE 會故意混為一談。

另外亂數原理也全都跳過 < 重點是亂數的產生原理也不只一種 >。

另本文附程式碼，不附執行結果，有興趣自己跑一遍。

最後請注意本文在區間表達裡，開區間與閉區間 括號的使用，也就是，

[a, b]  ,   (a, b]  , [a, b) ,  (a, b)

這四個表示的意義不同，

1. 基本使用

C/C++ 之亂數函式放在 stdlib.h / cstdlib 裡面，在使用時直接呼叫 rand() 便可。以下範例為產生 5 個亂數，並輸出。

**Code Snippet**

1. #include <stdio.h>
2. #include <stdlib.h>
3. int main()
4. {
5. int i;
6. for(i=0; i<5; ++i)
7. printf("%d ", rand());
8. getchar();
9. return 0;
10. }

 2. 亂數種子

將上述的程式多執行幾次會發現，怎麼每次亂數產生的都一樣？原因是沒設亂數種子。

那什麼叫亂數種子？

原理我不講了 < 因目的是要 "會" 用就好 >，簡單的說產生器是一組公式，公式要給「初始值」。

再怎麼給亂數這組公式一個初始值？用 srand( ) 。

那初始值該給多少？初始值給固定的值都沒用，要會隨著環境變動的值才有意義，

像是 記憶體使用量、process id 、CPU 使用率 等，這些都是會隨環境變動，

但有些變動性可能不大，而最常用來給初始值的，是時間，所以上述程式改如下。

**Code Snippet**

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <time.h>
4. int main()
5. {
6. int i;
7. unsigned seed;
8. seed = (unsigned)time(NULL); // 取得時間序列
9. srand(seed); // 以時間序列當亂數種子
10. for(i=0; i<5; ++i)
11. printf("%d ", rand());
12. getchar();
13. return 0;
14. }

而在 srand 那段，常常有人這麼寫

srand( (unsigned)time(NULL) );

這樣就不需暫存 seed 變數。

注意，srand 正常而言一份程式碼(專案)只能執行一次，如果它放在 for loop 裡，每次進行 rand 前就用 srand，會發現每次取出來的亂數是同一個數字。

3. 得知亂數最大值

後面會講為什麼要知道亂數最大值，這是一個重要的值。

C/C++ 提供的 rand() ，它有範圍限制，最小是 0 ，最大是多少？

最大被定義在 stdlib.h / cstdlib 裡面的 RAND\_MAX，所以要得知最大是多少的話

**Code Snippet**

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

目前可以確定的是，RAND\_MAX 至少會是 32767，最大會是多少不一定。但以筆者手邊的 Visual C++ 2010 環境而言，這個值是 32767。實際上 VC6.0 , VC2002 / 2003 , VC2008, VC2010 , gcc, Dev-C++ , Code::Blocks (with mingw) ，這個值也都剛好是 32767，只是他們實作的亂數細節不同而已。至於日後其他改版會不會讓 RAND\_MAX 更大？那就看那些軟體( compiler ) 如何實作了。

4. 產生固定範圍的整數亂數

我們以擲骰子為例，一個骰子有 6 個面，點數分別為1~6，要隨機擲一顆骰子怎麼做？

首先，1~6 剛好有 6 個數字，所以可以這麼寫

result = rand() % 6

% 叫取模運算子，不懂的話回去翻書。這樣下來可以確定，result 只有 {0, 1, 2, 3, 4, 5} 6 種可能而已。但實際上骰子的範圍是  1~6，而不是 0~5，怎麼辦？很簡單，只要把結果 + 1 就行了。原本的結果是 0~5 ，加1後結果變成 1~6。

總合以上說明，事實上我們可以給出一組公式，若要產生 [low, up]  之整數亂數，我們可以這麼做

rand() % (up - low + 1) + low

Q1 : 為什麼是 % (up-low+1) , 而不是 % (up-low) ?  
A1 : 因 low~up 一共有 (up-low+1) 個數。拿產生 [1,6] 來講，實際上共有 6-1+1 = 6 個數。

Q2 : 為什麼要加上 low ?  
A2 : 不加 low 的話實際上產生的是 [0 , up-low]，加上 low 的話才是 [low, up]。

Ex 1 : 模擬擲一顆骰子擲 10 次，並輸出其結果。

**Code Snippet**

1. #include <stdio.h>
2. #include <time.h>
3. #include <stdlib.h> // RAND\_MAX
4. int main()
5. {
6. int i;
7. srand( (unsigned)time(NULL));
8. for(i=0; i<10 ; ++i){
9. printf("%d ", rand() % 6 + 1);
10. }
11. getchar();
12. return 0;
13. }

有些數字可能不會出現 < 因為也才擲十次而已 >，但多執行幾次應會出現，且範圍一定是 1~6 。

Ex 2 : 摸擬擲 3 顆骰子 500 次，紀錄點數和出現的次數，最後輸出每個點數共出現幾次。

3 顆骰子點數最小為 3 ，最大為 18，所以輸出時只要判斷 3~18 出現的次數即可。下面程式碼沒優化過，對初學者而言較易懂。

**Code Snippet**

1. #include <stdio.h>
2. #include <time.h>
3. #include <stdlib.h> // RAND\_MAX
4. int main()
5. {
6. int RunTimes = 500  ; // 測試次數
7. int SumTimes[20]={0}; // 紀錄點數出現的次數, 全歸零
8. int i, sum, rnd;
10. // 進行測試
11. for(i=0; i<RunTimes; ++i) {// 測試 RunTimes 次
12. rnd = rand() % 6 + 1 ; // 第一顆骰子出現點數
13. sum = rnd;             // 紀錄總合
14. rnd = rand() % 6 + 1 ; // 第二顆骰子出現點數
15. sum = sum + rnd;       // 紀錄總合
16. rnd = rand() % 6 + 1 ; // 第三顆骰子出現點數
17. sum = sum + rnd;       // 紀錄總合
19. // 將出現 sum 點數之次數加1
20. SumTimes[sum] = SumTimes[sum]+1;
21. }
23. // 輸出結果
24. sum = 0;
25. for(i=3; i<=18; ++i) {
26. printf(" %2d 點出現了 %3d 次\n", i, SumTimes[i]);
27. sum = sum + SumTimes[i]; // 再驗證總合是不是500次
28. }
29. printf("共  %3d 次 \n", sum);
30. getchar();
31. return 0;
32. }

這裡要提醒，如果亂數產生的範圍已經超過 RAND\_MAX 的話，如產生 [-1000, +50000] 之亂數，必須額外進行處理，這種撰寫，只有前面的 RAND\_MAX 數字有機會出現，其他後面的數字全都沒機會看到。

另使用 % 取亂數，個人覺得較不妥，原因在 6. 再談整數亂數 說明並給方法。

5. 產生浮點數亂數

這裡是個重點，請別認為用不到很無聊跳過 < 如果熟的話大概也不會看這篇文章了吧。>

產生浮點數亂數，通常都是先取得  [0, 1) 之浮點數亂數 ( 可以包含零，但不包含 1 )。

這怎麼產生？還記得 RAND\_MAX 是什麼意思吧？是 rand() 可能產生的最大值，

所以寫出這段碼出來。

(double) rand() / (RAND\_MAX + 1.0 );

Q1 : 為什麼要特別在 rand() 前面轉型成 double ?  
A1 : 簡單的說，我怕有人雞婆，把後面的 1.0 自己寫成 1 ，這時候不加上 (double) 的話結果除出來一定是 0 ；若後面的 1.0 都不動它的話，前面的 double 可以拿掉無誤。

Q2 :  那為什麼分母還要特別加上 1.0 ?  
A2  : 前面有說過了，rand() 最大值可到 RAND\_MAX， 不加上 1.0 的話會使得 (double) rand() / RAND\_MAX ，結果有機率變成 1 ，但這與我的前提：不包含 1 是相違的。

Q3 : 那除了加上 1.0 這數字外，可以改成加其他數字啊！諸如 2.0 , 100.0, 10000.0 之類的。  
A3 : 又如我剛剛所說，是要產生 [0,1) 之間的浮點數亂數，假定 RAND\_MAX = 32767，如果加上 10000.0 的話，這個結果最大值會變成了 32767 / (32767+10000) = 0.247，明顯 [0.25, 1.0) 都沒機會生成了。但如果改成 0.5 , 之類，小於 1 較大的小數，到是可接受，不過這種數字幾乎沒人在用。

那，產生出 [low, up) 之浮點數隨機亂數(不含 up )怎做？

剛剛已給出了 rndf = [0, 1) 之公式，所以要擴展到 [low, up) 時，只要做點修改就行，

概念是 [low, up) 亂數，等於 (low~up 距離) \* ( [0,1) 亂數 ) + (下限 low)。

double low = 5.1 , up = 7.3, rndf, result;

rndf = (double) rand() / (RAND\_MAX + 1.0); // 產生 [0, 1) 浮點亂數

result = (up - low) \* rndf + low;  // 產生 [low, up) 浮點亂數

寫成一行型式

double low = 5.1 , up = 7.3, result;

result = (up - low) \* rand() / (RAND\_MAX + 1.0) + low;

Q4 : 上面範例都是在討論不含上界的情況，如果要含上界的話呢？  
A4 : 很簡單，把上面的 RAND\_MAX + 1.0 部份，全都改成 RAND\_MAX 即可，這樣就有機會出現上界。

6. 再談整數亂數

再回到擲骰子的問題上，要產生 [1, 6] 之間的整數亂數，事實上有另一種方法，就是先產生 [1, 7) 的浮點數亂數，之後再強制轉型成整數，所以程式碼如下所示。

**Code Snippet**

1. #include <stdio.h>
2. #include <time.h>
3. #include <stdlib.h>
4. int main()
5. {
6. int result;
7. double r01, rnd;
9. // 亂數種子
10. srand((unsigned)time(NULL));
11. // 產生 [0,1) 之亂數
12. r01 = (double)(rand()) / (RAND\_MAX + 1.0) ;
13. // 產生 [1,7) 之亂數
14. rnd = r01 \* (7.0 - 1.0) + 1.0;
15. // 強制轉型給 result
16. result = (int)(rnd);
17. printf("result = %d\n", result); // 輸出結果
18. getchar();
19. return 0;
20. }

Q1 : 為什麼是產生 [1, 7) 之浮點數亂數，而不是產生 [1,6] 之浮點數亂數？  
A1 : 重點在後半段還要強制轉型。若一開始就產生 [1, 6] 之浮點亂數時，要使得轉型後結果為 6 只有一種條件可達成：rand() 必須是 RAND\_MAX。這部份原理很簡單，但建議自己想想比較有收獲。

根據以上之敘述，觀查可納出一結論：當要產生出 [low, up] 之整數亂數時，可有另一種方式，

便是產生 [low, up+1) 之浮點數亂數後，再進行強制轉型成整數。如下。

**Code Snippet**

1. int low = -5 , up = 10 ; // 上下限
2. int result; // 結果
3. double r01 , r ;
4. r01 = (double)rand() / (RAND\_MAX+1.0); // 產生 [0, 1) 浮點亂數
5. r = r01 \* (up - low + 1.0) + low ; // 產生 [low, up+1) 浮點亂數
6. result = (int)r;  // 最後強制轉型。

甚至可包成副函式 或寫成一行。

**Code Snippet**

1. // 產生 [low, up] 之隨機整數亂數
2. int rand\_int(int low, int up)
3. {
4. return (int)((rand() / (RAND\_MAX+1.0)) \* (up - low + 1.0) + low);
5. }

如果是要產生 [low, up) 之隨機整數亂數的話呢？這在做陣列索引很常見，

因陣列有 N 個元素，範圍只能是 [0, N) ，而不能是 [0, N]。

實際上產生 [low, up) 之整數亂數，就是產生 [low, up-1] 之整數亂數，

一個方法是直接以 rand\_int(low, up-1) 方式代入上式；

硬要從函式裡面改的話，就是先產生 [low, up) 之浮點亂數後，

再強制轉型成整數資料型態。

**Code Snippet**

1. // 產生 [low, up) 之隨機亂數
2. int rand\_int2(int low, int up)
3. {
4. return (int)((rand() / (RAND\_MAX+1.0)) \* (up - low) + low);
5. }

接下來可以認真討論，為什麼大多數較不建議用取模運算子 (mod , %) 來求浮點亂數了。

我們先假設一種情況，若某個亂數產生器，他的 RAND\_MAX = 13，目前要產生 [0,3] 之整數亂數。

以取模運算子撰之， **rst = rand() % 4**，看一下數值分佈的情況。

rand() = 0, 4, 8, 12  : rst = 0  
rand() = 1, 5, 9, 13  : rst = 1  
rand() = 2, 6, 10      : rst = 2  
rand() = 3, 7, 11      : rst = 3

所以 rst = 2 與 rst = 3 出現的機率比較低，

機率比較低的 rst ，都被安排到 rst 可能出現之值的後半段。

再考慮

rst = (int)((rand() / (RAND\_MAX+1.0)) \* (up - low + 1.0) + low);

**rst = (int)( rand() / 14.0 \* 4 ) ;**

用乘、除法的情況

rand() = 0, 1, 2, 3    : rst = 0  
rand() = 4, 5, 6        : rst = 1  
rand() = 7, 8, 9, 10  : rst = 2  
rand() = 11, 12, 13  : rst = 3

所以 rst = 1 和 rst = 3 出現的機率比較低，

機率比較低的 rst ，都被均勻打散到 rst 可能出現之值範圍內。

鑑於亂數應符合均勻之特性，故較多人建議別用取模 (mod) 方式取整數亂數。

一樣的議題，若欲產生的整數亂數範圍超過 RAND\_MAX 時，這種方法也是有些數字沒辦法產生到。只是這種方法沒辦法產生的數字，是被打散到各區塊裡，而不是像取模運算子全擠在後半段。總之就是建議要額外處理。

7.  不均勻亂數問題

現假設一種情況是，希望不是每個數出現的機率都一樣，假設有 4 個數，

1 出現機率為 0.4 ;  2 出現機率為 0.1 ;

3 出現機率為 0.3 ;  4 出現機率為 0.2 ;

怎麼做？

針對這種較簡單的機率數字，1 2 3 4 出現的比率為 4 : 1 : 3 : 2，加總為 10，

所以有種做法如下

(1) 開大小為 10 的陣列 Arr[10]，

(2) 依序填入 4 個 1、1個2、3個3、2個4。

(3) 隨機產生 [0, 9] 之整數亂數 , pos，再取得 Arr[pos] 出來即可。

概念上之程式碼約如下述 < 這只是一份示例，會有更好的寫法 >。

**Code Snippet**

1. #include <stdio.h>
2. #include <time.h>
3. #include <stdlib.h>
5. int main()
6. {
7. int i, j, pos;
8. int Arr[10]; // 開大小為 10 的陣列 Arr[10]
10. //  依序填入 4 個 1、1個2、3個3、2個4
11. Arr[0]=Arr[1]=Arr[2]=Arr[3] = 1; // 4 個 1
12. Arr[4]=2 ; // 1 個 2
13. Arr[5] = Arr[6] = Arr[7] = 3 ; // 3 個 3
14. Arr[8] = Arr[9] = 4; // 2 個 4
16. srand( (unsigned) time(NULL) );
17. // 隨機產生 [0, 9] 之整數亂數 , pos，再取得 Arr[pos] 出來
18. for(i=0; i<10; ++i) {// 取 10 次
19. // 產生 [0,9] 整數亂數 pos
20. pos = (int)(rand() / (RAND\_MAX+1.0) \* 10) ;
21. // 取出 Array[pos]
22. printf("%d ", Arr[pos]);
23. }
24. getchar();
25. return 0;
26. }

[HomeWork] 依上述的數字出現之機率，做 10萬 次測試，最後真正實際上1, 2, 3, 4 出現之次數、機率為何？是否接近於當初設定之機率？

試再想另一種情形，若

10 出現機率為 0.123， 20 出現機率為  0.234，

30 出現機率為 0.345， 40 出現機率為  0.298，

< 加起來剛好等於 1 沒錯 >

照上面的方法，不就要設一個大小為 1000 的陣列了嗎？

那如果小數點後面加到 10 位數，不就要設一個大小為 10^10 的陣列了？

這個記憶體根本就放不下。

另一種方式是用累計機率，我們先做累計機率的表出來

[0] 10 : 機率 = 0.123 ，累計機率 = 0.123，令為 CP[0]

[1] 20 : 機率 = 0.234 ，累計機率 = 0.123 + 0.234 = 0.357，令為 CP[1]

[2] 30 : 機率 = 0.345，累計機率 = 0.357 + 0.345 = 0.702，令為 CP[2]

[3] 40 : 機率 = 0.298，累計機率 = 0.702 + 0.298 = 1.000，令為 CP[3]

累計機率算出來之後，我們只需要產生 [0, 1) 之随機浮點數亂數 rndf，

去檢查 rndf 落在哪段區間，rndf < CP[i] 之最小 i 即為所求。

程式碼示意如下。

**Code Snippet**

1. #include <stdio.h>
2. #include <time.h>
3. #include <stdlib.h>
5. int main()
6. {
7. int i, pos, n=4; // 4 個元素
8. int Num[4] = {10, 20, 30, 40}; // 欲出現之數字
9. double Prob[4]= {0.123, 0.234, 0.345, 0.298}; // 數字對應之出現機率
10. double CP[4];
11. double rf; // 隨機機率
13. srand( (unsigned)time(NULL));
14. // step 1 : 做累計機率計算
15. CP[0] = Prob[0];
16. for(i=1; i<n; ++i)
17. CP[i] = CP[i-1] + Prob[i];
19. for(i=0; i<10; ++i) { // 做 10 次測試
20. rf = rand() / (RAND\_MAX + 1.0) ; // 產生 [0, 1) 亂數
22. for(pos=0; pos < n; ++pos) // 查詢所在區間
23. if(rf <= CP[pos]) break;
24. printf("%d ", Num[pos]); // 輸出數字
25. }
26. getchar();
27. return 0;
28. }

 筆者所知只有這兩種方法，有其他方法歡迎討論。

不均勻亂數還有許多特殊的狀況，遇到時建議再念念機率統計，若是已有的機率模型，必可找到現有符合該機率模型之亂數產生器(像 tr1, boost , c++11 都有了) ，否則，只能從較特殊、列出來的機率模型那裡下手。

 8. 不重覆亂數問題 < 暴力法 >

要產生 20 個 [1,100] 不重覆之亂數，怎麼做？

一種作法是先開大小為 20 的陣列 Arr[20]，每產生一個亂數的時候，就到 Arr 裡面看有沒有重覆，如果沒有重覆才加進去，有重覆的話就再取下一個亂數。示例碼如下 < 贅變數很多，像 find 是可以完全拿掉的 >。

**Code Snippet**

1. #include <stdio.h>
2. #include <time.h>
3. #include <stdlib.h>
5. int main()
6. {
7. int n = 20; // 找 20 個相異亂數
8. int i, cnt, num, Arr[20];
9. int find;
10. srand( (unsigned)time(NULL));
11. cnt = 0; // 已有不重覆亂數之個數
12. while(cnt < n){
14. // 產生 [1, 100] 之整數亂數
15. // rand() / (RAND\_MAX+1.0)) \* (up - low + 1.0) + low
16. // num = rand()/(RAND\_MAX+1.0)\*(100-1+1.0) + 1;
17. num = (int)( rand() /(RAND\_MAX+1.0)\*100.0 + 1);
19. // 到 Arr 裡查有沒有重覆產生
20. find = 0; // 假設沒發現
21. for(i=0; i<cnt; ++i){
22. if(Arr[i]==num) { // 有發現
23. find = 1;
24. break;
25. }
26. }
27. //
28. if(find==0) {     // 真的沒發現
29. Arr[cnt]=num; // 加入 Arr 裡
30. ++cnt;        // 找到個數 +1
31. }
32. }
33. // 最後輸出
34. for(i=0; i<n; ++i){
35. if(i%10==0) puts("");
36. printf("%3d ", Arr[i]);
37. }
38. getchar();
39. return 0;
40. }

 上面這段碼可以正確跑出結果無誤。

這種方式大多被納為暴力法之一種模式，但實質上在某些情況它是蠻適合用的。如果只是要用二、三個相異的亂數，這方法很適合，直接用 do-while 做，甚至不需要開陣列就可完成。

剛剛的範例是，[1,100]，100 個數，挑 20 個相異亂數。但若把條件改過：

[1,32767]，挑32767個不重覆亂數，它的執行時間就頗費時了，這時就不考慮使用這方法。

9. 不重覆亂數問題 < 洗牌法 >

回到最初的問題，從 [1,100] 裡挑出 20 個不重覆之亂數，結果填到 Array 裡。這裡我們先為這些數字做點符號定義表示。

從 [low, up] 裡，挑出 n 個不重覆之亂數，結果填到 Array 裡。

洗牌 (shuffle) 法的概念是，剛剛的 [low, up] ，每個數字都視為撲克牌裡的一張牌，所以這副撲克牌共有 (up-low+1) 張，於是開陣列 Poker[up-low+1]，並填入 1: 100。

再來是模擬洗牌的過程，洗牌方式非常非常多！第一種是，隨機抽出第 pos1 張，再隨機抽出第 pos2 張，再將這兩張牌交換。進行 low-up+1 (100) 次。整個動作做完後，再把 poker 前面的 20 (n, 欲取幾個亂數) 張牌，放到 Arr 裡面，就是答案了。

看碼最清楚。

**Code Snippet**

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <time.h>
5. void shuffle\_1(int \*arr, int n, int low, int up)
6. {
7. int i, pos1, pos2, tmp;
8. int Size = up-low + 1; // 整份 poker 大小
10. // 配置一份 poker[Size]
11. int \* Poker = (int\*)malloc(sizeof(int) \* Size);
12. for(i=0 ; i<Size; ++i) // 填入 low~up
13. Poker[i] = i+low;
14. // 開始洗牌
15. for(i=0; i<Size; ++i){
16. // 隨機取出兩張 [0,Size) 之 poker
17. pos1 = (int)(rand() / (RAND\_MAX+1.0) \* Size);
18. pos2 = (int)(rand() / (RAND\_MAX+1.0) \* Size);
19. // 交換這兩張牌
20. tmp = Poker[pos1];
21. Poker[pos1] = Poker[pos2];
22. Poker[pos2]=tmp;
23. }
24. // 洗完牌, 前面的 n 張再給 Arr
25. for(i=0; i<n; ++i)
26. arr[i] = Poker[i];
27. free(Poker); // 釋放 poker
29. }
30. int main()
31. {
32. int low = 1, up=100;
33. int i, n = 20;
34. int arr[20];
35. srand((unsigned)time(NULL));
36. shuffle\_1(arr, n, low, up); // 洗牌
37. for(i=0; i<n; ++i) // 顯示結果
38. printf("%d ", arr[i]);
39. getchar();
40. return 0;
41. }

接下來就是細節了。目前流傳的洗牌方式有幾項

(1) shuffle\_1 : 隨機取出第 pos1 張、pos2 張，再進行交換，也就是上面的方式。

(2) shuffle\_2 : 為確保每張牌至少被換過一次，依序拿第 i 張牌出來，隨機取出第 pos1 張牌，第 i 張牌與第 pos 張牌交換。

故關鍵程式碼換如下。

**Code Snippet**

1. // 開始洗牌
2. for(i=0; i<Size; ++i){
3. // 隨機取出 [0,Size) 之 poker
4. pos = (int)(rand() / (RAND\_MAX+1.0) \* Size);
5. // 交換這兩張牌
6. tmp = Poker[pos];
7. Poker[pos] = Poker[i];
8. Poker[i]=tmp;
9. }

有幾個議題曾被討論過：(1) 多洗幾次牌是不是會比較亂？ (2) 最佳的洗牌次數是洗幾次？

筆者不知道上面這兩問題的答案。因這兩種洗牌方式沒被經過證明怎麼洗比較「亂」，

真正經過證明「怎麼洗較好」的是楊氏洗牌法 ( 或稱 Knuth Shuffle )。

Knuth Shuffle 在洗牌的過程重點在於：

(1) 從後面洗回來。for (j=size-1 ; j>0 ; --j) 注意，判別式裡沒有等於零。

(2) 取整數亂數 pos，範圍為 [0, j] ，交換 poker[j], poker[pos]

程式碼約如下述。

**Code Snippet**

1. void KnuthShuffle(int \*arr, int n, int low, int up)
2. {
3. int i, pos1, pos2, tmp;
4. int Size = up-low + 1; // 整份 poker 大小
6. // 配置一份 poker[Size]
7. int \* Poker = (int\*)malloc(sizeof(int) \* Size);
8. for(i=0 ; i<Size; ++i) // 填入 low~up
9. Poker[i] = i+low;
10. // 開始洗牌
11. for(i=Size-1; i>0; --i){
12. // 隨機取出 [0, i] 之 poker
13. pos = (int)(rand() / (RAND\_MAX+1.0) \* (i+1));
14. // 交換這兩張牌
15. tmp = Poker[pos];
16. Poker[pos] = Poker[i];
17. Poker[i]=tmp;
18. }
19. // 洗完牌, 前面的 n 張再給 Arr
20. for(i=0; i<n; ++i)
21. arr[i] = Poker[i];
22. free(Poker); // 釋放 poker
23. }

使用 shuffle 必須額外再多配置一份 (up-low+1) 之記憶體空間，若本身 poker 張數很多 ( 欲挑選的範圍很大)，但欲取得的值很小 ( n 很小 ) ，事實上也不適合用 shuffle，除了浪費空間之外，還浪費了一開始填數字的時間，此時反而以暴力法來做較為恰當。像是在 [1,20000] 取出 10 個相異亂數時，此時用暴力法便較為恰當。

10. 不重覆亂數問題 < 排序法 >

再續上個問題，從 [1,100] 裡挑出 20 個不重覆之亂數，結果填到 Array 裡。

1~100 有 100 個元素，排序法方式是直接開兩個陣列 : int Rst[100], int Rnd[100]，

Rst[100] 從 1 填到 100，Rnd[100] 是連續取100個亂數填進去，

填完之後，對 Rnd 做排序，而在排序過程中有用到交換，Swap (Rnd[i], Rnd[j])

交換時連 Rst 也一起交換 Swap(Rst[i], Rst[j])，程式碼約如下述 < 排序法用較低效之排序 > 。

**Code Snippet**

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <time.h>
5. #define SWAP(a,b){int t=a; a=b; b=t;}
6. void SortShuffle(int \*arr, int n, int low, int up)
7. {
8. int i, j;
9. int Size = (up-low+1);
10. int \*Rst = (int\*)malloc(sizeof(int) \* Size);
11. int \*Rnd = (int\*)malloc(sizeof(int) \* Size);
13. if(Rst==NULL || Rnd==NULL) return;
15. for(i=0; i<Size; ++i){
16. Rst[i] = low + i; // 依序填入數值到 Rst
17. Rnd[i] = rand();  // 對 Rnd 取亂數
18. }
19. // 對 Rnd 做排序
20. for(i=0; i<Size-1; ++i){
21. for(j=i+1; j<Size; ++j){
22. if(Rnd[i] > Rnd[j]) {
23. // 交換時連 Rst 也一起交換
24. SWAP(Rnd[i],Rnd[j]);
25. SWAP(Rst[i], Rst[j]);
26. }
27. }
28. }
29. // Rst 前 n 筆存入 arr
30. for(i=0; i<n; ++i)
31. arr[i] = Rst[i];
32. // 釋放記憶體
33. free(Rnd), free(Rst);
34. }
35. int main()
36. {
37. int low = 1, up=100;
38. int i, n = 20;
39. int arr[20];
40. srand((unsigned)time(NULL));
41. SortShuffle(arr, n, low, up); // 洗牌
42. for(i=0; i<n; ++i) // 顯示結果
43. printf("%d ", arr[i]);
44. getchar();
45. return 0;
46. }

這種作法較少人用。原因是它記憶體空間比其他方法至少多出兩倍，另外時間也大多花在排序法上面 (較佳也是 nlogn 複雜度)，故幾乎沒人用。

常見的不重覆亂數解決方案，大致上就這三種。

11. 大亂數問題 (I)

大亂數問題在上面有先提過了，假設要產生的整數亂數範圍是 [0, 50000]，或產生的浮點數亂數精度為 1e-6，怎麼處理？

先講講整數亂數 [0, 50000]，倘若 RAND\_MAX 只到 32767 時，以 % 方式而言，無論怎麼產生，[0,32767] 可正常產生，但 [32768,50000] ，共 17234 個數完全產生不了。即使先產生 [0,1) 浮點數，即用 rand() / RAND\_MAX 這方式，也一樣會有 17233 個數產生不了，只是這 17233 個數不是最後的那幾個，而是被均勻打散到 [0,50000] 裡面而已。

再來看浮點數亂數，要達到1e-6 精度問題，這只是相同問題換個型態出現而已。若 RAND\_MAX 只到 32767，產生的亂數最小精度是 1/32767，約為 3.1 E -5 ，達不到精度要求為 1e-6 之需求。

最快徹底解決這問題的方式，是直接換一套亂數產生器的函式庫，這些在 C++11 , boost, tr1 裡面已有非常豐富，甚至也有專門在寫亂數函式庫的 library，甚至較有水準的數值分析函式庫也大多會有較佳品質的亂數函式庫出現。拿到時注意幾個點：RAND\_MAX 是多少？若以浮點亂數出現的話，其精度是多少？還有亂數重覆周期是多少。更重要的是，注意他們的亂數函式庫支不支援多行緒？最好找支援多行緒的函式庫，未來移植才比較沒問題。這幾點很重要。

一種勉強可接受 ( 其實也是大多還不太會用 library 之 coder 的解決方案 ) 之方式為：一次取兩個亂數，將數值擴大。假設 RAND\_MAX = 32767，佔了 15 bits( 111111111111111(2) = 32767(10) )，試考慮以下程式碼。

**int** high = rand() << 15;

**int** low = rand();

**int** rst = high | low;

甚至三行可寫成一行

   int rst = ( rand() << 15 ) | rand();

如此下來可產生 30 bits 之亂數，範圍從 [0, 215-1] 變成了 [0, 230-1]。若有需要，可再取二次、取三次、取四次等等，但這會有潛在問題存在，一方面使用 rand() 之亂數產生器，通常週期並不非常長 ( 像 vc, gcc 之 rand 週期只到 231 左右)，且平均度也有待測試，這也是筆者建議直接再找另一支亂數產生器之原因。

12. 大亂數問題 (II)

**現假設一問題為，該如何產生  [0, 1] 之間，10-12 精度之浮點亂數產生器。**

這部份只是簡單的數學推導，已知該怎麼做的可略過不看。

step 1 : 計算 所需 NEW\_RAND\_MAX

假設 RAND\_MAX = 32767 (15 bits) , 先想想一般的亂數精度可以怎麼求

double precision = 1.0 / RAND\_MAX = 1.0 / 32767 = 3.05 \* 10-5

所以要達到 10-12 精度時

10-12 = 1.0 / NEW\_RAND\_MAX  
NEW\_RAND\_MAX = 1012

組合出的亂數最大值至少要 1012 才可滿足。

step 2 :  從 NEW\_RAND\_MAX 計算所需 bits 數

以無號數二進位而言，n bits 可表達之最大數為 2n-1 ，故可列下以下不等式

2n-1 >= 1012  ，忽略 1 所帶來之影響，兩邊取 log10  
log10(2n) >= 12，  
n log10(2) >= 12  
n >= 12 / log10(2) = 39.86

由於 n 必為大於等於1之整數，故取 40。

step 3 : 由 bit 數產生亂數

在假設 RAND\_MAX = 32767 之情況下，取一次 rand() 有 15 bits，故要到 40 bits 至少要取 3 次才可達到。但以筆者手邊環境而言，int / unsigned int 只有 32 bits ，沒辦法達到 40 bits 之要求，故改用資料型態 unsigned long long ( 更好的做法是用 uint64\_t ) 去存結果，下面是一種作法。

**Code Snippet**

1. typedef unsigned long long u64;       // typedef
2. u64 rst = ( (u64)(rand()) << 25 ) |   // bit[39:25]
3. (u64)(rand()) << 10 ) |           // bit[24:10]
4. (u64)(rand()& 0x3ffULL ) ;        // bit[9:0]

切割方式為 15 + 15 + 10 = 40 bits，左移 bits 數依序為 [15+10, 10, 0]。 但考慮到高位元之循環率較低位元循環率小，所以將 40 切割成 14 + 13 + 13，且取出時取高 bits 為主，依序應該左移 bits 數為 [13+13,13,0]。

**Code Snippet**

1. typedef unsigned long long u64; // typedef
2. u64 rst = ( \
3. ( (u64)(rand() >> 1) << 26 ) | // high 14, L shift 26bits
4. ( (u64)(rand() >> 2) << 13 ) | // high 13, L shift 13bits
5. ( (u64)(rand() >> 2)       )); // high 13, L shift 0bits

 結束之後，這只能產生 [0,240-1] 之亂數產生器，要再產生 [0,1] 之浮點亂數，就再除上 240-1。

**Code Snippet**

1. typedef unsigned long long u64; // typedef
3. u64 rand40() {
4. return ( \
5. ( (u64)(rand() >> 1) << 26 ) | // high 14, L shift 26bits
6. ( (u64)(rand() >> 2) << 13 ) | // high 13, L shift 13bits
7. ( (u64)(rand() >> 2)       )); // high 13, L shift 0bits
8. }
10. double randf40() {
11. const u64 NEW\_RAND\_MAX = (1ULL << 40) - 1ULL;
12. return (double)rand40() / NEW\_RAND\_MAX;
13. }

 大亂數問題至此結束。提醒，一般簡單統計用的亂數可以用此法產生沒錯 ( 像一些演化式演算法，或蒙地卡羅演算法)，若用於加解密等，通常不會再用 rand() 方式進行亂數產生。

 13. 其他

亂數其他議題相當多，有些也不好實作出來，本篇所提是較為基礎之部份，其他諸如 蒙地卡邏 MAMC、其他亂數分佈等議題，便不於此文探討。

亂數是程式語言學習上的一個重點，學校的程式課程多半會教、出練習題，

因此也是新手常遇到的問題。

在此列出一點C語言中亂數使用上的心得以及常犯錯誤

[1] 入門用法

srand( time(NULL) );

for( i = 0; i < 10; i++ )

printf( "Random number #%d: %d\n", i, rand() );

呼叫 srand()函式將亂數初始化，可以 time(NULL) 作為初始種子(seed)，

或自行設定任意數。不先呼叫本函數、或種子值固定，都會造成新手常見

的「每次執行產生的亂數結果都相同」問題。

[2] 產生固定範圍的亂數

# 0 ~ n-1

一般常見的寫法有：

1. rand() % n;

2. (int)( n \* ( rand() / (float)(RAND\_MAX+1) ) );

兩種寫法n的上限皆不能大於RAND\_MAX，否則某些範圍內的數字永遠不會出現。(Why?)

一般C語言的RAND\_MAX只有32767，於使用上需特別注意。

第1種寫法在n有點大時( RAND\_MAX/k < n <=RAND\_MAX, say k < 5 )可能會出現亂數分佈不均。因此以第2種為佳。

# a ~ b

將上面的寫法改一下，變成 a + ( 0 ~ (b-a+1) - 1 )即可。

也就是上面的 n 變成 b-a+1。

==

[3] 產生更大範圍的整數亂數

使用系統內建的rand()上限一般只到 32767，使用上不方便，會有前述的問題。

改善方法有很多，這邊列出一種 RAND\_MAX = 32767 時的簡單改寫法：

int rand31() {

// RAND31\_MAX = 2147483647

return ( (rand() << 16) + (rand() << 1) + (rand() & 1) );

}

unsigned int rand32() {

// RAND32\_MAX = 4294967295

return ( (rand() << 17) + (rand() << 2) + (rand() & 3) );

}

分別可產生32位元的有號及無號正整數亂數。

==

[4] 產生 0 到 小於1 的浮點數

上面的例子用到的 rand() / (float)(RAND\_MAX+1)

即可產生0~小於1的浮點數，但是如上所述，應用範圍只能到RAND\_MAX。

RAND\_MAX=32767簡單的改寫法如下，應用範圍提高到 16777215，是32位元浮點數極限：

float rand24() {

// RAND24\_MAX = 16777215

return ( (rand() << 9) + (rand() >> 6) ) / (float)0x01000000;

}

[例] 產生在選定範圍均勻分佈的浮點亂數

# 產生 0 ~ 7777777的整數亂數： n = (int)( 7777778 \* rand24() )

# 產生 -2.5 ~ 2.5 的浮點亂數： x = ( rand24() - 1 ) \* 5

# 產生 a ~ b 的浮點亂數： x = rand24() \* ( b - a ) + a

==

EX: 如何使用 rand() produces -3.0到 3.0之值

Range=6. Assume precision is 小數點後面是two digits

因要取的區間是 6, 所以先取區間: 600

rand()%601 … produce 0~600

(rand()%601)/100 產生 0.00~6.00

**(float)(rand()%601)/100)-3.0 產生 -3.00~3.00**

[5] 產生特殊分佈的亂數

這邊我們以 frand()函數 表示 範圍為0~小於1的浮點數 uniform random number

frand() 可以是上例的rand24() 或其他方式得到的亂數

# 高斯分佈(常態分佈)亂數 [**http://en.wikipedia.org/wiki/Normal\_distribution**](http://en.wikipedia.org/wiki/Normal_distribution)

\*產生平均值μ=0 標準差平方σ^2=0 的分佈

Gaussian0\_rand = sqrt( -2 \* log( frand() ) ) \* sin( PI \* frand() )

使用2次frand()亂數產生高斯亂數。需要注意的是當第1個 frand() = 0 時，

會出現log(0)的數學錯誤，因此在使用上需做修正。可以用if來判斷，或是

將 log( frand() ) 改寫成 log( frand() + min\_float )

min\_float 是指 32bits 或 64bits 浮點數中最小的正浮點數。

\*產生平均值μ 標準差σ的分佈

Gaussian\_rand = Gaussian0\_rand \* σ + μ

==

# 其他分佈

如果要做出特殊形狀分佈的亂數，只要那個形狀你有辦法積分，「大概」就可以

「簡單」的從 uniform rand# 轉成該函數的形狀

│f(x) p b

│ ◢◣ ∫ f(x)dx = frand() \* ∫ f(x)dx

│ ◢◣◢██◣ a a

│ ◢██████◣

├──────────> 解方程式得到 p = g( frand() )

a p b

p就是新的亂數，其分佈應該是如你所願。

[例] f(x) = C ， 從a積到b。 解： p = frand() \* (b-a) + a (是否似曾相識！！)

[例] f(x) = x ， 從a積到b。 解： p = sqrt( frand() \* (b^2 - a^2) + a^2 )

==

以上的簡單介紹相信在一般作業上的使用已經足夠。

**由於內建的亂數是偽亂數，可預測性高，**

**萬萬別使用在對於安全性或隨機性要求很高的商業行為或是學術研究**

有需求的人再自行去研究吧

[**http://en.wikipedia.org/wiki/Random\_number\_generation**](http://en.wikipedia.org/wiki/Random_number_generation)

另外，RAND\_MAX的值隨著編譯器的種類會有很大差別，使用上述擴大亂數範圍

的簡單函數時記得先確認自己RAND\_MAX的值再加以改寫，以免出現不明錯誤....