迴圈控制架構設計
12 Days of Christmas 程式 與 Spiral Tap 程式 的迴圈設計

丁培毅

迴圈的語法

每一個迴圈都包括**四個**部份:
A. 初始
B. 迴圈執行條件判斷
C. 迴圈主體
D. 迴圈變數增減

1. for 迴圈

```
for (i=0; i<10; i++)
{
    printf("%d", i);
}
```

2. while 迴圈

```
i=0;
while (i<10)
{
    printf("%d", i);
    i++;
}
```

3. do while 迴圈

```
do
{
    printf("Input a positive number");
    scanf("%d", &x);
}
while (x<=0);
```


什麼時候該使用迴圈設計程式?

- 知道迴圈的語法以後, 看到程式裡使用迴圈時, 可以瞭解**程式在做什麼**, 可是這樣子還不太夠…
- 常常會覺得不曉得別人怎麼弄出那樣子的迴圈? 看起來沒有那麼直接, 有一點衝動想用**平鋪直敘**的方式寫就好了, 什麼事情要做兩次就把程式拷貝一次就是了!
- 不是不認得這幾個語法, 也不是機制太複雜, 就算有一些陷阱也還不是太大的問題, 問題在**什麼時候該用迴圈**? 該怎麼看到**重複性的架構**? 使用迴圈語法以後要達到什麼**目的**才算是充分地使用?

使用時機 (1/8)

1. 需要重複做很多次相同的事情時

例如:
A. 重複輸入 5 個整數、計算並列印它們的 3 次方
B. 列印 5 次 "hello world\n"
C. 重複執行 x = x + y 5 次

- 也就是如果不使用迴圈你需要寫出下面的程式

```
scanf("%d", &x); printf("%d\n", x*x*x);
scanf("%d", &x); printf("%d\n", x*x*x);
scanf("%d", &x); printf("%d\n", x*x*x);
scanf("%d", &x); printf("%d\n", x*x*x);
scanf("%d", &x); printf("%d\n", x*x*x);
```

```
printf("hello world\n");
printf("hello world\n");
printf("hello world\n");
printf("hello world\n");
printf("hello world\n");
```

```
x = x + y;
x = x + y;
x = x + y;
x = x + y;
x = x + y;
```

使用時機 (2/8)

- **複製貼上**程式碼一般來說是**盡量避免**的事情, 不是因為程式需要比較炫, 用比較多的語法, 而是因為重複的程式碼在維護時很快就會發生**一致性**的問題, 修改其中一個忘了另一個
- 使用**迴圈**

```
for (i=0; i<5; i++)
{
   scanf("%d", &x); printf("%d\n", x*x*x);
}
```

```
for (i=0; i<5; i++)
   printf("hello world\n");
```

```
for (i=0; i<5; i++)
   x = x + y;
```

- 使用迴圈語法撰寫**得到的好處**包括:
  A. 程式碼短一點
  B. 修改的時候只要改一遍, 不會造成一致性的問題
  C. 程式容易調整, 既然重複出現了 2 次, 也許有一天會需要改成 3 次, 那麼只需要改一個數字就好
  D. 甚至可以讓使用者輸入要重複幾次, 不需要設計成固定的

使用時機 (3/8)

2. 需要重複做很多次類似的事情時

例如:

A. 需要重複輸入多對整數 (a, b) 並且分別印出 a+b, a-b, a*b, a/b, a+b, a-b, a*b, a/b, ...
B. 需要列印出 "1\n", "3\n", "5\n", "7\n" ...
C. 需要把 26 個整數變數加總起來

- 兩次以上

```
scanf("%d%d", &a, &b); printf("%d\n", a+b);
scanf("%d%d", &a, &b); printf("%d\n", a-b);
scanf("%d%d", &a, &b); printf("%d\n", a*b);
scanf("%d%d", &a, &b); printf("%d\n", a/b);
   ...
```

```
printf("1\n");
printf("3\n");
printf("5\n");
...
printf("99\n");
```

```
x = x + a;
x = x + b;
x = x + c;
...
x = x + z;
```

- 也許覺得上面這樣子寫沒什麼不對的, 每一列都不太一樣啊! 迴圈主體裡面不是都長一樣的嗎??? 既然不一樣那不就要跟迴圈說掰掰了?!

使用時機 (4/8)

- **抽象化**: **這是 科學 / 工程 / 藝術 / 生活 上很重要的能力**
  - 你**搭乘公車**的時候都要從某一站上車, 給某個金額的車資, 到某一站下車; 你**買早餐**的時候都要去某家店給付出某個金額買某一種餐點; 你要去**上課**都要在某個時間帶某一本書去某一棟大樓的某一間教室..
  - 每次搭公車都不太一樣吧? 每天買早餐都不太一樣吧? 每節上課都不太一樣吧?
  - 可是搭公車你會看成是同一件事, 買早餐會看成是同一件事, 上課會看成是同一件事, 都是每天會重複做的, 你心裡想的就是每天重複做**一樣**的事情吧, 不太會覺得每天都像做完全不同的事, 要用不一樣的方法來處理吧!

只專注於某一部份, 不在意的部份視而不見
只看到一致的地方, 忽略不一致的地方
就是**抽象化**的能力

使用時機 (5/8)

- 數學、物理、化學這些科目裡你看到很多公式, 這些都是從實際發生的現象裡把一些不在意的東西忽略掉, 剩下共同的核心
- 那麼**不太一樣的地方**該怎麼辦呢??? 如果是日常生活的話, 可能就有一些小規則, 比如說如果坐基隆市公車要付多少錢, 坐基隆客運由某站到某站要付多少錢, 坐台汽由某站到某站要付多少錢.... 那就是要**判斷**了.... **if** ..... 就出現了, 如果比較規律化的話, 可以運用變數、陣列、迴圈、或是函式的語法再進一步簡化

```
scanf("%d%d", &a, &b); printf("%d\n", a+b);
scanf("%d%d", &a, &b); printf("%d\n", a-b);
scanf("%d%d", &a, &b); printf("%d\n", a*b);
scanf("%d%d", &a, &b); printf("%d\n", a/b);
   ...
```

```
printf("1\n");
printf("3\n");
printf("5\n");
...
printf("99\n");
```

```
x = x + a;
x = x + b;
x = x + c;
...
x = x + z;
```

- **概念上** 這些還是可以看成是**重複的程式碼**, 它們重複的部份是:
  A. 輸入整數 (a, b), 印出 a 和 b **某種**算術運算的結果
  B. 列印**某個**規律化遞增的整數序列資料
  C. 把**某個**資料加總到變數 x 裡

使用時機 (6/8)

- 藉由**迴圈語法**以及**迴圈控制變數**來實現**重複執行多次的概念**

```
for (i=0; i<次數; i++)
{
   某件事情
}
```

- 不太一樣的地方藉由 **if 敘述**以及**迴圈控制變數**來判斷

```
for (i=0; i<20; i++)
{
   scanf("%d%d", &a, &b);
   if (i%4==0)
      printf("%d\n", a+b);
   else if (i%4==1)
      printf("%d\n", a-b);
   else if (i%4==2)
      printf("%d\n", a*b);
   else if (i%4==3)
      printf("%d\n", a/b);
}
```

```
for (i=0; i<50; i++)
   printf("%d\n", 2*i+1);
```

```
for (i=0; i<26; i++)
   switch (i)
   {
   case 0: x = x + a; break;
   case 1: x = x + b; break;
      ...
   case 25: x = x + z; break;
   }
```


使用時機 (7/8)

- 第二個程式片段是很好的, 類似而重複的部份中有變化的用變數的運算 **2*i+1** 來描述, 很簡潔!!
- 第三個程式片段一定是不好的, 用了 26 個變數, 還用很大的一個 switch 敘述, 這是因為我們還沒有講如何使用**陣列變數**, 如果資料不是放在 a, b, c, ..., z 的變數裡, 而是放在 data[0], data[1], ..., data[25] 這 26 個變數裡面, 程式就可以簡化成

```
int data[26];
   …
for (i=0; i<26; i++)
   x = x + data[i];
```

這時候所有有變化的地方完全用變數 **i** 以及陣列變數 **data[i]** 的存取來描述, 很簡潔!!

使用時機 (8/8)

- 第一個程式片段勉強可以, 可以進一步用 **函式** 簡化

```
for (i=0; i<20; i++)
{
   scanf("%d%d", &a, &b);
   printArithmetic(i%4, a, b);
}
void printArithmetic(int type, int a, int b)
{
   if (type==0)
      printf("%d\n", a+b);
   else if (type==1)
      printf("%d\n", a-b);
   else if (type==2)
      printf("%d\n", a*b);
   else if (type==3)
      printf("%d\n", a/b);
}
```

- 迴圈內的敘述現在變成幾乎完全一樣了, 差異用 **i%4** 來描述, 在**概念上**簡化了迴圈的表達, 但是程式碼其實幾乎是一樣的
- 進一步可以透過函式指標陣列來簡化, 概念上有點像接下來我們要介紹的常數字串指標陣列配合迴圈的應用

12 Days of Christmas 歌詞列印程式裡的迴圈設計

The 12 Days of Christmas

On the **first** day of Christmas
my true love sent to me:
A Partridge in a Pear Tree

On the **second** day of Christmas
my true love sent to me:
2 Turtle Doves
and a Partridge in a Pear Tree

On the **third** day of Christmas
my true love sent to me:
3 French Hens
2 Turtle Doves
and a Partridge in a Pear Tree

On the **fourth** day of Christmas
my true love sent to me:
4 Calling Birds
3 French Hens
2 Turtle Doves
and a Partridge in a Pear Tree

On the **fifth** day of Christmas
my true love sent to me:
5 Golden Rings
4 Calling Birds
3 French Hens
2 Turtle Doves
and a Partridge in a Pear Tree

On the **sixth** day of Christmas
my true love sent to me:
6 Geese a Laying
5 Golden Rings
4 Calling Birds
3 French Hens
2 Turtle Doves
and a Partridge in a Pear Tree

On the **seventh** day of Christmas
my true love sent to me:
7 Swans a Swimming
6 Geese a Laying
5 Golden Rings
4 Calling Birds
3 French Hens
2 Turtle Doves
and a Partridge in a Pear Tree

On the **eighth** day of Christmas
my true love sent to me:
8 Maids a Milking
7 Swans a Swimming
6 Geese a Laying
5 Golden Rings
4 Calling Birds
3 French Hens
2 Turtle Doves
and a Partridge in a Pear Tree

On the **ninth** day of Christmas
my true love sent to me:
9 Ladies Dancing
8 Maids a Milking
7 Swans a Swimming
6 Geese a Laying
5 Golden Rings
4 Calling Birds
3 French Hens
2 Turtle Doves
and a Partridge in a Pe

On the **tenth** day of Christmas
my true love sent to me:
10 Lords a Leaping
9 Ladies Dancing
8 Maids a Milking
7 Swans a Swimming
6 Geese a Laying
5 Golden Rings
4 Calling Birds
3 French Hens
2 Turtle Doves
and a Partridge in a P

On the **eleventh** day of Christmas
my true love sent to me:
11 Pipers Piping
10 Lords a Leapin
9 Ladies Dancing
8 Maids a Milkin
7 Swans a Swimn
6 Geese a Laying
5 Golden Rings
4 Calling Birds
3 French Hens
2 Turtle Doves
and a Partridge i

On the **twelfth** day of Christmas
my true love sent to me:
12 Drummers Drumming
11 Pipers Piping
10 Lords a Leaping
9 Ladies Dancing
8 Maids a Milking
7 Swans a Swimming
6 Geese a Laying
5 Golden Rings
4 Calling Birds
3 French Hens
2 Turtle Doves
and a Partridge in a Pear Tree

需要用迴圈?! 有重複嗎?!

- 這樣也有看到重複的東西?? @@
- 有啊! 每一段都是 On the .... 是有重複, 但是每一段裡面好像變動還蠻大的
- 人很厲害 (至少受過訓練的工程師是這樣), 都很會抽象化, 會特地把不太一樣的東西看成是一樣的, 目的就是**簡化**, 把歌詞寫成下面這樣就覺得重複性大多了吧!

  On the xxx day of Christmas my true love sent to me: .
  On the xxx day of Christmas my true love sent to me: ..
  On the xxx day of Christmas my true love sent to me: ...

- 如果先不看冒號後面的 … 的話, 根本就是一樣的, 當然應該用迴圈來設計

```
for (i=0; i<3; i++)
   printf("On the xxx day of Christmas\nmy true love sent to me:\n");
```


運用迴圈語法設計 (2/4)

- 實際上也不是 xxx, 而是 "first", "second", "third" 常數字串, 所以我們配合常數字元指標陣列來設計, 迴圈裡面幾乎是一樣的東西, 變動的部份用陣列變數來處理

```
const char *numbers[] = {"first", "second", "third"};
for (i=0; i<3; i++)
   printf("On the %s day of Christmas\nmy true love sent to me:\n“, numbers[i]);
```

- 上面我假裝沒看到冒號後面的 **…**, 其實是用 **.** 代表一列歌詞, **..** 代表兩列歌詞, **...** 代表三列歌詞, 明明每一列歌詞都不一樣, 但是我暫時把它當成是一樣的, 這樣子簡化的話程式會變成

```
const char *numbers[] = {"first", "second", "third"};
for (i=0; i<3; i++) {
   printf("On the %s day of Christmas\nmy true love sent to me:\n", numbers[i]);
   for (j=0; j<i+1; j++) printf(".");
   printf("\n");
}
```

運用迴圈語法設計 (3/4)

- 接下來要把 … 變成不一樣的東西, 還是先簡化一點, 用數字 **1** 代表 A Partridge …, **2** 代表 2 Turtle …

On the first day of Christmas my true love sent to me: **1**
On the second day of Christmas my true love sent to me: **21**
On the third day of Christmas my true love sent to me: **321**

程式只需要改一點點而已

```
const char *numbers[] = {"first", "second", "third"};
for (i=0; i<3; i++) {
   printf("On the %s day of Christmas\nmy true love sent to me:\n", numbers[i]);
   for (j=0; j<i+1; j++) printf("%d", i+1-j);
   printf("\n");
}
```


運用迴圈語法設計 (4/4)

- 接下來換成不同字串的輸出

On the first day of Christmas my true love sent to me:
A Partridge

On the second day of Christmas my true love sent to me:
2 Turtle
A Partridge

On the third day of Christmas my true love sent to me:
3 French
2 Turtle
A Partridge

程式需要再一次運用常數字元指標陣列來設計迴圈

```
const char *numbers[] = {"first", "second", "third"};
const char *lines[] = {"A Partridge", "2 Turtle", "3 French"};
for (i=0; i<3; i++) {
   printf("On the %s day of Christmas\nmy true love sent to me:\n", numbers[i]);
   for (j=0; j<i+1; j++) printf("%d", lines[i-j]);
   printf("\n");
}
```

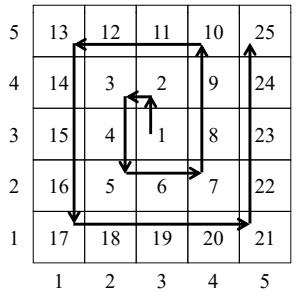
- 最後處理 A Partridge 和 and a Partridge 的變化

- 提醒你一下, 開學到這個時候大概是第六週了, 有些同學開始覺得力不從心, 開始覺得程式轉來轉去變來變去的, 快要受不了程式執行結果永遠是不正確的了, 甚至認真思考是不是自己不適合學這些東西!! 手腳快的已經開始準備轉系、準備重考了!
- 不曉從這幾個範例還有『撰寫程序式程式的基本方法與要求』那裡有沒有發現關鍵….在於 **題目是可以修改** 的…別笑太大聲, 這好像是公平的**考試制度**下一直很直覺排除掉的方法
- **「華生，你終於突破盲點了」!**

螺旋狀數字排列 (Spiral Tap) 程式裡的迴圈設計

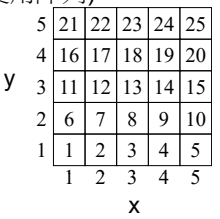
問題

- 下圖的方格狀棋盤有 n 行、 n 列, n 為奇數, 圖中橫座標和縱座標都是由 1 開始
- 由棋盤的正中央開始以螺旋狀方式逆時針順序排列 1 到 n^2 這些整數
- 請撰寫一個程式,
  輸入一整數 n 代表棋盤的寬度,
  輸入另一整數 t 代表目標數字,
  程式計算出 t 的橫座標與縱座標值



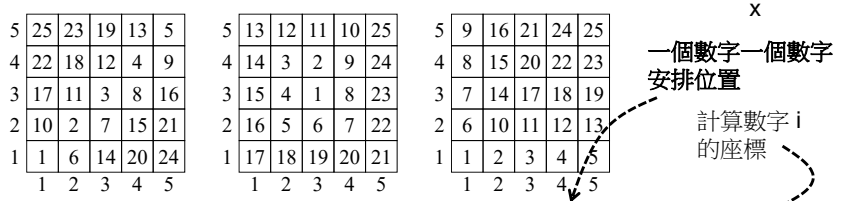
簡化與聯想

- 在課程裡還沒有正式介紹到**陣列**, 所以下面的說明裡我們先**不要**使用**陣列**的語法 (當 n 很大時也不該使用陣列)
- **目標**: 找到數字 target 對應的座標 (x, y)
- 考慮右圖這個簡化的問題



簡單公式:
```
y = (target-1) / 5 + 1;   // "target" 在哪一列
x = (target-1) % 5 + 1; // "target" 在哪一行
```

- 比較一般化的位置安排



一個數字一個數字安排位置

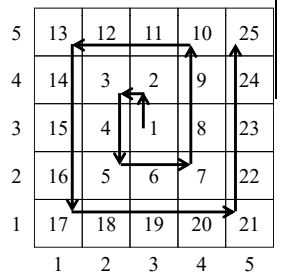
計算數字 i 的座標

- 推導公式比較麻煩一些, 讓我們用『**模擬(Simulation)**』的方法來稍微暴力一點處理這個問題

```
for (i=2, x=y=1; i<=target; i++) {
    x = x+1;
    if (x==6) x=1, y = y+1;
}
```


嘗試設計

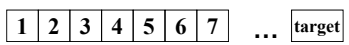
- 基本方法: **模擬 (simulation)**
- **迴圈**語法: **重複**做**類似**的事
- 在這個問題裡**模擬安排每個數字的位置**時看到重複出現的動作嗎? **尋找規律性**



1:(3,3)
2:(3,4)
3:(2,4)
4:(2,3)
5:(2,2)
6:(3,2)
7:(4,2)
8:(4,3)
9:(4,4)
10:(4,5)
11:(3,5)
12:(2,5)
...

由 **2:(3,4)** 開始安排 **target-1** 個數字的位置
每一步 **把數字加 1, 計算新數字的座標 (x,y)**

- 如果是一維方格, 計算每個數字的座標就容易了!!



```
for (i=2,x=1; i<=target; i++)
    x = x+1;
```

- 修改座標要依據方向
- 每一步要注意是否轉向

版本一

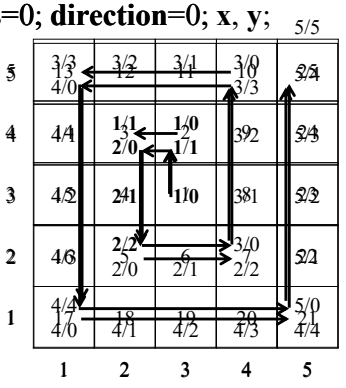
- **while 迴圈直接由 1, 2, 3, … 往上計數到 target**
  每一直線段的長度 1, 1, 2, 2, 3, 3, 4, 4, 5, …

檢查是否 **target** <= n*n **path length / steps in each path**

```
int value=1, pathLen=1, step=0, nTurns=0, direction=0, x, y;
x = y = n/2 + 1;
while (value < target) {
    if (step == pathLen) {
        direction = (direction+1)%4;
        pathLen+=nTurns%2;  nTurns++;
        step=0;
    }
    step++;
    value++;
    nextCoordinate(&x, &y, direction);
}
```

Directions: **0:** 上 **1:** 左 **2:** 下 **3:** 右

下一次迴圈的資料



其實不需要用 nTurns%2, 直接用 direction 也可以判斷 pathLen 需不需要加 1

版本一 (續)

- 使用函式語法

```
void nextCoordinate(int *xPtr, int *yPtr, int direction)
{
   switch (direction)
   {
   case 0: /* 上 */
      *yPtr = *yPtr + 1;
      break;
   case 1: /* 左 */
      *xPtr = *xPtr - 1;
      break;
   …
   }
}
```

direction:
0: 上
1: 左
2: 下
3: 右

- 不使用函式 (用下列敘述取代前一頁 nextCoordinate(&x,&y,direction))
  - 使用 **if** 語法

```
if (direction==0) y++;
else if (direction==1) x--;
else if (direction==2) y--;
else if (direction==3) x++;
```

  - 不使用 **if** 或是 **switch** 語法

```
x += (direction-2) % 2;
y += (1-direction) % 2;
```

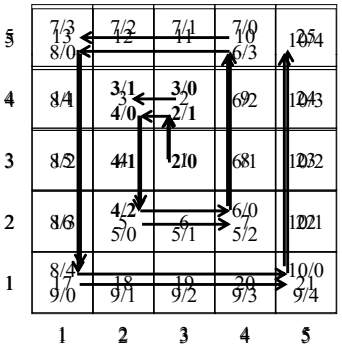
還不清楚 int * 用法的話, 請使用「不使用函式」的版本

版本一 (簡化)

- **while** 迴圈直接由 **1, 2, 3, …往上計數到 target**
  每一直線段的長度 1, 1, 2, 2, 3, 3, 4, 4, 5, …

檢查是否 **target** <= n*n **÷2** **path length control / steps in each path**

```
int count=0; pathLenCtrl=2, step=0; direction=0; x, y;
x = y = n/2 + 1;
while (++count < target)
{
   nextCoordinate(&x, &y, direction);
   if (++step == pathLenCtrl/2)
   {
      direction = (direction+1)%4;
      pathLenCtrl++, step=0;
   }
}
```



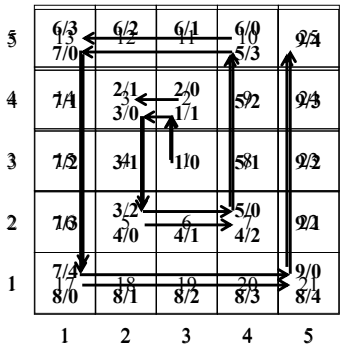
Directions: **0:** 上 **1:** 左 **2:** 下 **3:** 右

版本一 (換一種講法)

- **while** 迴圈直接由 **1, 2, 3, …** 往上計數到 **target**
  給每一直線段編號 1, 2, 3, 4, 5, 6, 7, 8, 9, …
  每一直線段的長度 1, 1, 2, 2, 3, 3, 4, 4, 5, …

檢查是否 **target** <= n*n **path index / steps in each path**

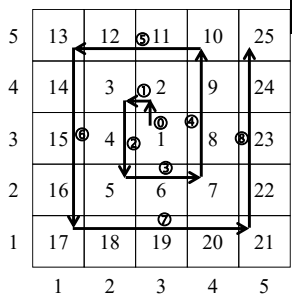
```
int count=0; pathIndex=1, step=0; direction=0; x, y;
x = y = n/2 + 1;
while (++count < target)
{
   nextCoordinate(&x, &y, direction);
   if (++step == (pathIndex+1)/2)
   {                    線段長度
      direction = (direction+1)%4;
      pathIndex++, step=0;
   }
}
```



Directions: **0:** 上 **1:** 左 **2:** 下 **3:** 右

版本二: 計數迴圈 (counting loop)

- 分段的計數迴圈
  - n=5
  - 總共 2*n-1=9 直線線段
    ⓪, ①: 長度 1; ②, ③: 長度 2; …



```
check if target <= n*n
value = 0;
x = y = n / 2 + 1;
for (i=0; i<2*n-1; i++)  // 第 i 個線段
   for (j=0; j<i/2+1; j++)  // 線段中第 j 步
   {
      if (++value == target) 輸出 x, y 並且結束 兩層的迴圈
      nextCoordinate(&x, &y, i%4);
   }
```

方向: **0:** 上 **1:** 左 **2:** 下 **3:** 右

直接 return 0

版本三: 加快執行速度

- 最內層 k=0, 開始和結束的數字都是 (2k+1)(2k+1)=1
- 第二層 k=1, 開始和結束的數字分別是 (2k-1)(2k-1)+1=2 和 (2k+1)(2k+1)=9
- 第 k 層的開始數字是 **(2k-1)(2k-1)+1** 結束數字是 **(2k+1)(2k+1)**
- 先用一個 for 迴圈找到滿足下式的 **k** 值
  (2k-1)(2k-1)<target<=(2k+1)(2k+1)
- count=(2*k-1)*(2*k-1); // 0~(k-1)層總數
  x = n/2+k+1, y=n/2+k+1; // 開始的座標
  for (i=1; i<=**4**; i++) { // 4 段直線
  for (j=0; j<**2*k**; j++) { // 每一段走 2k 步
  nextCoordinate(&x, &y, i%4);
  if (++count == target) 輸出 x, y 並且*結束*兩層的迴圈
  } 從 **10** 開始走時, 假想 **9** 在 (5,5), 由 9 開始往左走四步, 這樣子寫方向只
  } 要有四種就可以, 如果是從真正的位置 (4,4) 開始走, 就變成先要往上走

5	13	12	11	**10**	**25**
4	14	3	2	**9**	24
3	15	4	**1**	8	23
2	16	5	6	7	22
1	17	18	19	20	21
	1	2	3	4	5

版本四: 直接計算公式

- 先用一個 for 迴圈找到滿足下式的 **k** 值
  (2k-1)(2k-1)<target<=(2k+1)(2k+1)
- infimum =(2*k-1)*(2*k-1);
  x = n/2+k, y=n/2+k;
- steps = target – infimum;
- 最外圈的四段:
  0 1, 2, …, 2k,
  1 2k+1, 2k+2, …, 4k,
  2 4k+1, 4k+2, …, 6k,
  3 6k+1, 6k+2, …, 8k
  iseg = (steps-1) / (2*k);

5	13	12	11	10	**25**
4	14	3	2	**9**	24
3	15	4	**1**	8	23
2	16	5	6	7	22
1	17	18	19	20	21
	1	2	3	4	5

k=2
2k-1=3

- x-(steps-1) | x-(2k-1) \ x-(2k-1)+(steps-4k) / x+1
  y+1 | y+1-(steps-2k) \ y+1-2k / y+1-2k+(steps-6k)