計算 x^n

1. 請撰寫一個程式，計算 x^n, 其中 x 爲倍精準浮點數, n 爲整數
2. 程式輸入輸出範例如下：

```
Please input the value of x and n: 0.9 22
0.900^22=0.098477
Please input the value of x and n: 0.99 550
0.990^550=0.003975
Please input the value of x and n: 1.5 30
1.500^30=191751.059233
```

```
#include <math.h>
…
y = pow(x, n);
```

雖然使用這個 pow 函式可以很方便地計算倍精準浮點數的 x^n，甚至不是整數的 n 也可以計算
不過我們現在這個範例還是很重要的，它提供一個
1. 迴圈應用範例
2. 遞迴應用範例
3. 尾端遞迴應用範例

迭代法 (Iterative)

1. 計算 x^n 最直接的方法就是運用 for 迴圈乘 x 乘 n 次，如下：

```
int i; double exp=1;
for (i=0; i<n; i++) exp *= x;
```

如果 n 有 k 個位元, 運算量 $O(k^2 2^k)$

程式寫起來很簡單，但是很沒有效率 (需要 n 次乘法，實際上應該只需要 log n 次乘法)，對很大的 n 來說，甚至沒有辦法在有限時間內完成 (資訊安全的運算裡常需要處理 $n \sim 2^{1024}$)

2. 有效率的方法，計算範例：

$0.9^{1234} = .9^{(2^{10}+2^7+2^6+2^4+2)} = .9^2 \bullet .9^{2^4} \bullet .9^{2^6} \bullet .9^{2^7} \bullet .9^{2^{10}}$

$= .9^2 \bullet (((.9^2)^2)^2)^2 \bullet ((.9^{2^4})^2)^2 \bullet (.9^{2^6})^2 \bullet (((.9^{2^7})^2)^2)^2$

```
double exp, x=.9; int n=1234;
for (exp=1; n>0; x*=x,n/=2)
  if (n%2==1) exp *= x;
```

運算量 $O(k^3)$

n	1234	617	308	154	77	38	19	9	4	2	1	**0**
n%2	0	1	0	0	1	0	1	1	0	0	1	
x	.9	$.9^2$	$.9^4$	$.9^8$	$.9^{16}$	$.9^{32}$	$.9^{64}$	$.9^{128}$	$.9^{256}$	$.9^{512}$	$.9^{1024}$	
exp	1	$.9^2$	$.9^2$	$.9^2$	$.9^{2+16}$	$.9^{18}$	$.9^{18+64}$	$.9^{82+128}$	$.9^{210}$	$.9^{210}$	$.9^{210+1024}$	$.9^{\mathbf{1234}}$

2. 前面這個計算方法可以用下面這張圖說明它的精神 – **平方再乘**
如果以二進位表示 $n = b_0 \cdot 2^2 + b_1 \cdot 2 + b_2$

$$1 \cdot x^{b_2} \quad (x^{b_2}) \cdot (x^2)^{b_1} \quad (x^{b_2} \cdot (x^2)^{b_1}) \cdot (x^4)^{b_0}$$

平方 乘 平方 乘

3. 另外一種很接近的寫法如下，精神也是 – **平方再乘**

$$x^{b_0} \xrightarrow[\text{平方}]{} (x^{b_0})^2 \cdot x^{b_1} \xrightarrow[\text{乘}]{} (x^{2 \cdot b_0 + b_1})^2 \xrightarrow[\text{平方}]{} (x^{2 \cdot b_0 + b_1})^2 \cdot x^{b_2} \xrightarrow[\text{乘}]{}$$

寫成程式的時候需要多做一次迴圈來算出 n 二進位有幾位數

```
double exp=1, x=.9; int ord=1, n=1234, n1=n;    // x=.9, n=1234
while (n1>1) n1 /= 2, ord *= 2;                  // ord = 2^10
while (ord>0) {
  exp *= exp;
  if (n/ord==1) exp *= x;
  n%=ord; ord/=2;
}
```

運算量 $O(k^3)$

n	1234	210	210	210	82	18	18	2	2	2	**0**
n/ord		1	0	0	1	1	0	1	0	0	1
ord	1024	512	256	128	64	32	16	8	4	2	1
exp	1	$.9 .9^2$	$.9^4$	$.9^8$	$.9^9.9^{18}$	$.9^{19}.9^{38}$	$.9^{76}$	$.9^{77}.9^{154}$	$.9^{308}$	$.9^{616}$	$.9^{617}.\mathbf{9^{1234}}$

遞迴法 (Recursive)

1. 前面在迭代法裡面談的兩種計算方法，都可以寫成遞迴的函式，遞迴函式基本上就是一個自己呼叫自己的函式
2. **設計方法**步驟如下：
   a. 定義出遞迴函式以及其參數
   b. 描述清楚這個函式能夠解的問題
   c. 將所解的問題縮減爲較小的問題
   d. 用比較小的參數呼叫這個函式
   e. 問題縮到最小時可以直接寫出答案

運算量 $O(k^3)$

3. 方法一：
   a. double power(double x, int n)
   b. 呼叫這個函式，函式執行完畢傳回的數值就是 x^n
   c. $x^n = x^{n-1} \cdot x$；如果 n 是偶數，可以縮減得很快 $x^n = (x^2)^{n/2}$
   d. ❶ return power(x, n-1) * x ❷ return power(x*x, n/2)
   e. if (n==0) return 1;

4. 方法二：

   運算量 $O(k^3)$

   a. double power(double x, int n)

   b. 呼叫這個函式，函式執行完畢傳回的數值就是 x^n

   c. $x^n = x^{n-1} \cdot x$；如果 n 是偶數，可以縮減得很快 $x^n = x^{n/2} \cdot x^{n/2}$

   d. ❶ return power(x, n-1) * x ❷ y = power(x, n/2); return y * y;

   e. if (n==0) return 1;

5. 除了尾端遞迴的函式之外，所有運用遞迴設計的函式都需要仔細估計遞迴函式呼叫自己的次數 (例如 1000)，一般來說這個次數乘上函數參數與區域變數需要的記憶體位元組數 (例如 1000 * (8+4+8+小常數)) 需要小於系統堆疊空間 (大約 10^6 個位元組)，否則遞迴函式會把系統堆疊用光，使得程式錯誤或是當掉

6. 撰寫完後你通常需要把呼叫遞迴函式的回傳值、參數以及重要變數**表列**出來，可以檢查是否正確，例如上面方法二的遞迴：

x	.9	.9	.9	.9	.9	.9	.9	.9	.9	.9	.9	.9	.9	.9	.9	.9
n	1234	617	616	308	154	77	76	38	19	18	9	8	4	2	1	0
回傳	$.9^{1234}$	$.9^{617}$	$.9^{616}$	$.9^{308}$	$.9^{154}$	$.9^{77}$	$.9^{76}$	$.9^{38}$	$.9^{19}$	$.9^{18}$	$.9^{9}$	$.9^{8}$	$.9^{4}$	$.9^{2}$	$.9^{1}$	1

尾端遞迴函式

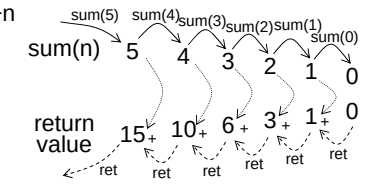
1. **遞迴函式**是函式裡面呼叫自己本身的函式

   尾端遞迴函式 (Tail Recursive Function, Tail Recursion) 也是一種遞迴函式，額外的要求是每次呼叫自己之後，必須立刻回傳結果，不能做任何其它事

   例如：計算 f(n) = 0 + 1 + 2 + … + n 的遞迴函式

一般遞迴函式

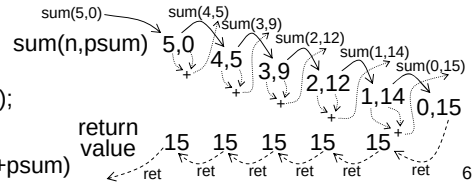
```
int sum(int n) {  // 計算 f(n)=0+1+2+…+n
   if (n==0)
      return 0;
   else
      return sum(n-1)+n;
}
```



C++ 預設參數

尾端遞迴函式

```
int sum(int n, int psum=0) { // 計算 f(n)=1+2+…+n+psum
   if (n==0)
      return psum;
   else
      return sum(n-1, n+psum);
}
```



計算 f(n)=1+2+…+(n-1)+(n+psum)

2. 尾端遞迴函式比一般的遞迴函式難寫難讀，爲什麼要寫這種尾端遞迴函式呢? 因爲 C++ 編譯器可以對這種函式最佳化，使得它就算呼叫自己非常多次，不會使用額外的堆疊空間 (因爲尾端遞迴函式在下一層函式回傳後並不需要使用上一層函式裡面的任何參數或是變數的數值，所以其實堆疊上的堆疊框可以直接在呼叫下一層函式時使用，上一層函式的 ret address，直接作爲下一層函式的 ret address，只需更換呼叫的引數值即可)

3. 撰寫計算 $f(x,n)=x^n$ 的尾端遞迴函式，設計方法如下：

   a. 分解問題 n爲奇數: $f(x,n)=prod \cdot x^n=(prod \cdot x) \cdot x^{n-1}$

   n爲偶數: $f(x,n)=prod \cdot x^n=prod \cdot (x^2)^{n/2}$ (方法二)

   ①一定有一個參數 prod 是部份答案，這個部份答案一層一層越來越接近要計算的 f(x,n)，開始時呼叫 powerTR(x,n,f(x,0))

   ②下層函式 f(x',n') 需要計算出原來的 f(x,n)，而不是子問題的答案，如此才能夠在每一層直接回傳最後結果，呼叫 powerTR(x',n',prod) 時部份答案就是執行函式所計算的和最後答案 f(x,n) 的剩餘差異，在呼叫 powerTR(x',0,prod) 時基本上沒有計算的動作，所以部份答案就是最後的答案 f(x,n)

   b. 根據分解的問題定義出遞迴函式以及其參數

   double powerTR(double x, int n, double prod=1)


```
double powerTR(double x, int n, double prod=1) {  // 計算 f(n)=prod · x^n
   if (n==0)
      return prod;
   else if (n%2) // n is odd
      return powerTR(x, n-1, prod*x); // 計算 f(n)=(prod · x) · x^(n-1)
   else // n is even
      return powerTR(x*x, n/2, prod); // 計算 f(n)=prod · (x^2)^(n/2)
}
```

範例：計算 0.9^{22} 之函式呼叫參數

.9000, 22,1
.8100,11,1
.8100,10, .8100
.6561, 5, .8100
.6561, 4, .5314
.4305, 2, .5314
.1853, 1, .5314
.1853, 0, .0985

4. 上面這個程式是依據「方法二」來設計的，請思考一下是否可依據「方法一」來設計尾端遞迴函式?

   分解問題 n爲奇數: $f(x,n)=prod \cdot x^n=(prod \cdot x) \cdot x^{n-1}$

   n爲偶數: $f(x,n)=prod \cdot x^n=prod \cdot x^{n/2} \cdot x^{n/2}$

   n 爲偶數時困難度比較高，計算 f(x,n) 時需要 f(x,n/2)，相當於在遞迴函式的呼叫參數需要紀錄 f(x,n-1), f(x,n-2), …, f(x,n/2)，當 n 值很大時沒有辦法如此實作