

1. 遞迴

① 大問題 → 小問題 → 解決

別人門牌

② 階乘

最大公因數

搜尋

費式數列

組合數

河內塔

fractal 碎形

③ binary search

用 divide and conquer strategy 分而擊之

④ `s.substr(size-1, 1)` 把最後一個字元一個拿出來

⑤ `void writeBackward (string s, int size) {`

`if (size > 0) {`

`cout << s.substr(size-1, 1);` ← 如果倒過來

`writeBackward (s, size-1);` ← output → c → a → t

`} // if`

`} // writeBackward ()`

ex: `s = cat` `size = 3`

1. `t` 2. `a` 3. `c` 4.
`s = 'cat'` `s = 'ca'` `s = 'c'` `s = ''`
`size = 3` `size = 2` `size = 1` `size = 0.`

Double A

⑥ $a > b$. 從 b 加到 a .

```
int sum(int a, int b) {
    if (a > b)
        return sum(a-1, b) + a;
    else
        return b;
} // sum()
```

⑦ 最大公因數

$gcd1(x, y) = x$ if $y = 0$
 $= gcd1(x, y \bmod x)$ if $y > x$
 $= gcd1(y, x \bmod y)$ otherwise

$gcd2(x, y) = y$ if $x \bmod y = 0$
 $= gcd2(y, x \bmod y)$ otherwise

```
int gcd1(int x, int y) {
    if (y == 0) return x;
    else if (y > x) return gcd1(x, y % x);
    else return gcd1(y, x % y);
} // gcd1()
```

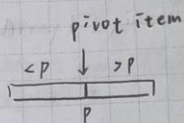
$int gcd2(int x, int y) \{ \rightarrow \text{有效效率}$
 if $!(x \% y)$ return y ;
 else return $gcd2(y, x \% y)$;

⑧ 在 array 中找
 1. 二分法
 2. 選擇矩陣

⑧ 在 array 中找第 k 小

1. 二分法

2. 選擇樞紐 (pivot item)



3. 找其中一邊 (類似二元搜尋)

⑨ 河內塔 $2^n - 1$

ch = 抽象化

物件導向

所有東西都是物件

① 封裝 hides inner details

② 繼承 reused

③ 多型

抽象化 → 模組化的延伸

ADT: Abstract Data Types. (ex: List)

class 達到封裝 (含 methods & data) $\begin{cases} \text{private} \\ \text{public} \end{cases}$

Constructors 用來存資料

Destructor 結束程式時, 刪除資料

別人門牌

Inheritance

class ColoredSphere: public Sphere

⊗ 父類別 Sphere

↓
⊗ 子: colorSphere

Private: only class instances

Protected: subclass instances

Public: any class instances

ch 3. 串列.

pointer 指標

陣列: 需移動資料

鏈結串列: 不需移動資料

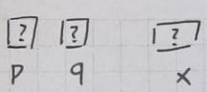
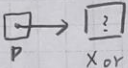
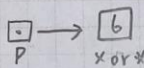
(int*) p; (不是 NULL)
(位址) 還沒有房子

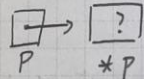

p = &x; 房子 x 的門牌

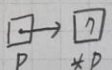
先 p = new int; 會有新房子

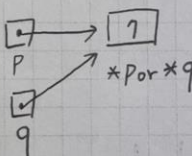
{ 一定要 delete p; 釋放記憶體

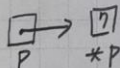

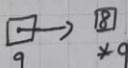
★ p = NULL;



(a) `int *p, *q;`  申請空白門牌
`int x;`
 (b) `p = &x;` } b 要先做才能做 c  抄寫別人門牌
 (c) `*p = 6;`  鳩佔



(d) `p = new int;`   緊急配置

(e) `*p = 7;`  堆放家當

(f) `q = p;`  抄寫至另一門牌

(g) `q = new int;`  
`*q = 8;` 

(h) `p = NULL;`   (不正確的寫法)

(i) `delete q;`  (正確).
`q = NULL;` 

動態陣列

```
int arraySize = 50;
```

```
double *anArray = new double[arraySize];
```

陣列名稱 = 指標

```
anArray[z] = *(anArray + z) 後移 z.
```

```
delete [] anArray; 刪掉一羣
```

```
struct Node {
```

```
    int item;
```

```
    Node *next;
```

```
} // Node
```

```
if head is NULL → empty
```

```
Node *p; 節點
```

```
p = new Node;
```

```
印內容 for (Node *cur = head; cur != NULL; cur = cur->next)
    cout << cur->item;
```

Ch 4 遞迴解題

中序式 $a+b$

前序式 $+ab$

後序式 $ab+$

不用知道先 \times 後 $+$

中 $a+b*c/d$

前 $+a/*bcd$

後 $abc*d/+$

中序運算式

$\langle infix \rangle = \langle identifier \rangle |$

$(\langle infix \rangle \langle operator \rangle \langle infix \rangle)$

$\langle operator \rangle = + | - | * | /$

$\langle identifier \rangle = a | b | \dots | z$

中: $(a+b)*c$ $((a+b)*c)$

前: $*+abc$

後: $ab+c*$

前序、後序優點 = 不用考慮優先權

前序: 一個前序式再接上「非空字串」一定不是前序式

Backtracking Problems

ex: 八皇后、迷宮、數字猜謎

遞迴 vs. 數學歸納法

① $if(n \leq 0)$

return 1;

else

return $n * fact(n-1)$

特性

- $fact(0) = 0! = 1$

- $fact(n) = n! = n(n-1)(n-2) \dots * 1$ 假設

$if(n > 0)$

- $fact(n+1) = (n+1) * fact(n) =$

$(n+1) * n!$

最簡

歸納

② 可評估效率

心得:

ch1:

大一計概用遞迴時,都覺得很麻煩,能用迴圈就用迴圈,現在比較了解遞迴才發現沒那麼複雜。

ch2:

大一用到 class 的時候,就只是用他而已,都沒有仔細了解他的功能、作用是什麼,現在終於知道也會運用了。

ch3:

大一計概用 pointer 的時候, delete 東西之後都沒有把變數 = NULL, 現在知道這是一件很重要的事情,而 delete 東西前也不要讓他先 = NULL, 會出錯

ch4:

以前都不知道有分中序、後序、前序,現在知道後發現後序、中序可以讓運算更順暢,不用考慮先 $*$ /後 $+$ -