單元1 遞迴

* 迴圈 v.s. 遞迴 → 鏡子的原理，較精簡

* Recursive Functions

階乘、最大公因數、搜尋、費氏數列、組合數、河內塔

1. binary search → divide and conquer 分而擊之
2. 倒印字串 → ① 字串減1 ② size > 0

```
void F (string s, int size) {
    if (size > 0) {
        cout << s.substr(size-1, 1);   輸出最後一個字元
        F(s, size-1);                  遞迴呼叫
    }
}
```

3. 最大公因數

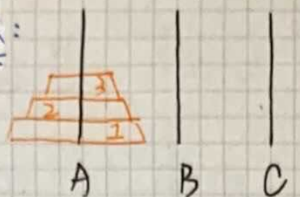
法1：
```
int gcd1 (int x, int y) {
    if (y==0) return x;
    else if (y>x) return gcd1(x, y%x);
    else          return gcd1(y, x%y);
}
```

法2：
```
int gcd2 (int x, int y) {
    if !(x%y) return y;
    else      return gcd2(y, x%y);
}
```

x=9, y=6
gcd(6,3)=3
x=6, y=3
gcd(3,0)=
x=3, y=0
return ③

4. 河内塔

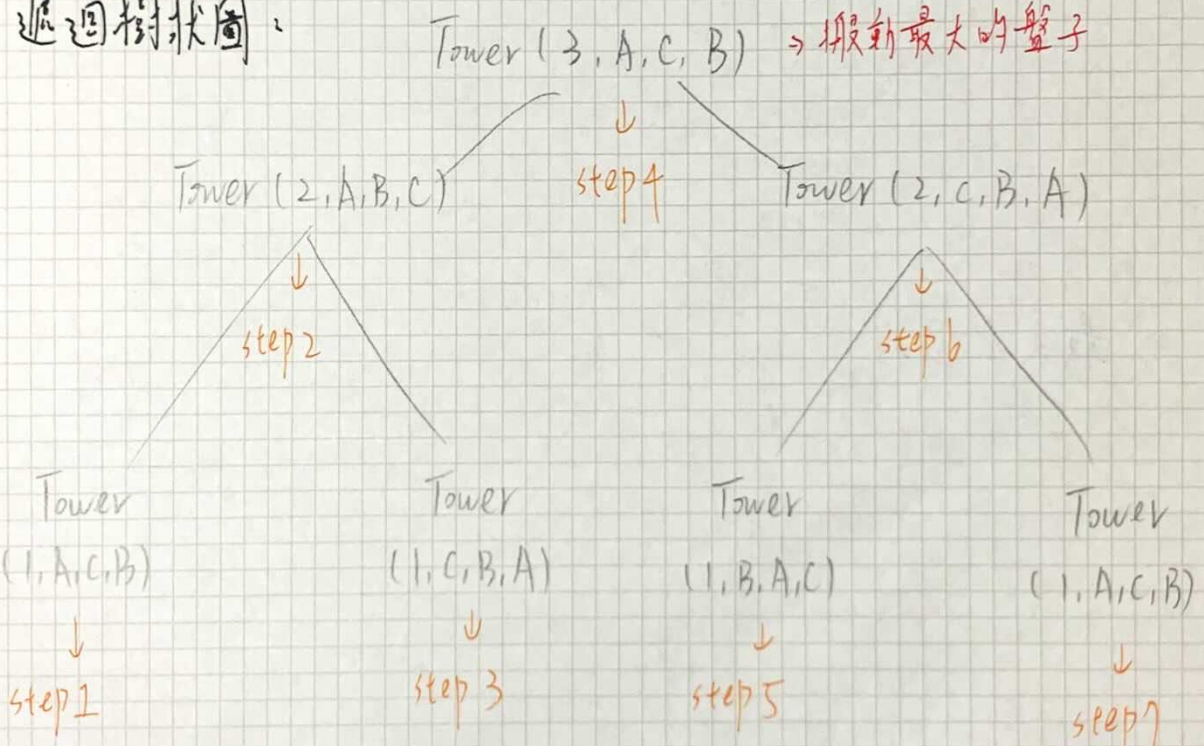
圖示：




```
Tower ( count , source , goal , spare ) {
    if (count == 1) Move a disk from source to goal
    else {
        Tower (count-1, source, spare, goal);
        Tower (1, source, goal, spare);
        Tower (count-1, spare, goal, source);
    }
```

（count 盤數，source 起始，goal 終點，spare 輔助）

遞迴樹狀圖：



單元2 抽象化

* simple ADT's : Grocery List, Appointment book, Sphere (球)
* Attributes (屬性) & Behaviors (運算)
* class of object (called instance)
  - → Attribute : data members
  - → Behaviors : methods
* 3個主要方法
  ① 封裝 ② 繼承 ③ 多載

繼承: ex: class Colors : public Sphere
父類別: Sphere
子類別: Colors → Colors是Sphere的一種.
(subclass)
* private: only class can use
  protect: subclass "
  public: all "

class Rational {
  protect:
    ...
  public:
    ...
};

多載: subclass用class的函式
(若連要取都相同稱「覆載」)

* 運算合約: Purpose, Assumptions, Input, output

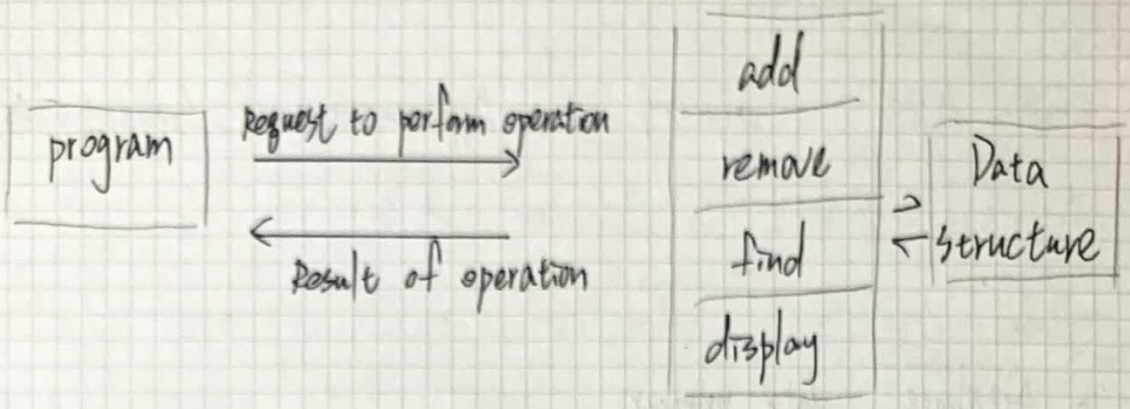
1. Modularity 模組化 ① 高內聚 ② 低耦合.
   每個函式僅量做單一件事且彼此不要回傳太多資訊
   (功能性抽象化: 描述&實作分開)

* ADT List Operations (Grocery List)

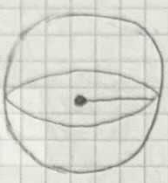
建構 - 解構 - 是否為空 - 計算個數 - 插入 - 刪除 - 檢索

aList.creatList() ... 建構aList
aList.insert(1, milk, success) ... 插入milk在位置1
aList.insert(2, eggs, success) ... 插入eggs在位置2.
aList.Remove(3, success) ... 刪除位置3的資料
  *後面位置資料前移.

* wall of ADT operations



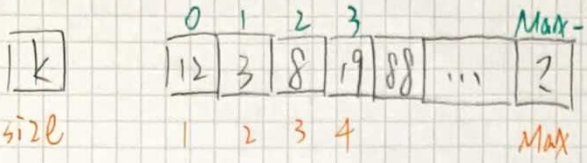
* class 的 constructor 概念 → class 在宣告時做設初事
預設建構



```
class Sphere( ) {
  public :
    Sphere( );   程式執行此後，會在記憶體建構一了空間存 Sphere
                 這個型別，就跟用 int 一樣
    :
  private :
    :
} // class
```

* Sphere :: Sphere( ) → 表示這了 class 中的這了 function
* classes 中能 destructor（解構），以免記憶體用太多
* Sphere :: Sphere( ) → 使用 class 裡的此函式
  Sphere :: Sphere (double initialRadius) → 使用 class 裡的此函式且有設定預設值

* An Array-Based ADT List

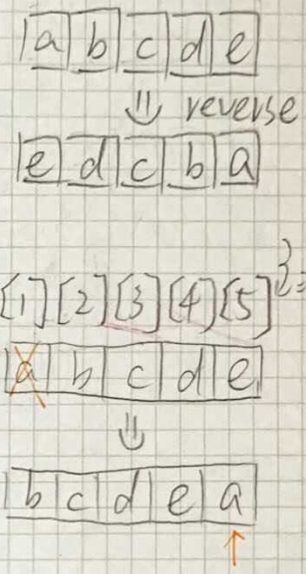


Array index → 底層實作：陣列索引

ADT List position → 高階描述：串列位置

* Reverse the entire list

```
Reverselist( in alist: List, out source: boolean)
    for (i=1 to alist.getlength()-1) {
抓出來  alist.retrieve(1, dataItem, success);   (1=位置)
刪掉    alist.remove(1, success);
插入    alist.insert(alist.getlength-i+2, ...);
```



單元7 鏈結串列

* Array → 需移動資料 / Linked list → 不需移動資料（動態陣列）

* 改掛勾：Pointer

指標＝門牌，(int *)P 表示指定的門牌號碼(P)指向一個大小為 int 的房子，在記憶体中就有一個門牌號碼(由P存)的空房子

```
int x;   // 配置一個新房子
P = &x;  // &x = 房子x的門牌
```

not same

指向別人而已 ← P = &x;

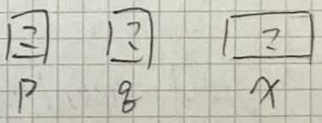
自己擁有房子 ← P = new int; // 申請一棟大小為 int 的新房子
(動態配置)
↳ 若沒此程序，P只是門牌，經過此程序後才會有自己的房子

- delete P; // 歸還房子
- P = NULL; // 徹底忘記門牌

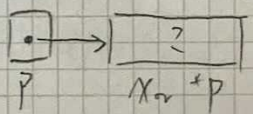
只有後者會 memory leak？(要黑洞)

more clear？

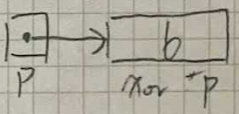
(a) 申請空白門牌 `int *P, *q;` `int x;`



(b) 抄寫別人門牌 `P = &x;`

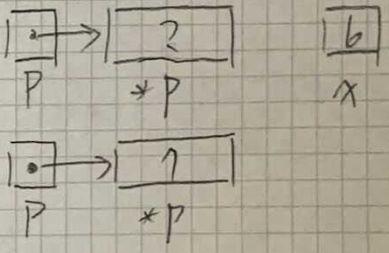


(c) 搞占鵲巢 `*P = 6;`



(d) 聚為配置 `P = new int;`

(e) 堆放家當 `*P = 7;`



順序不可調換

* 動態配置陣列

```
int arraysize = 50;
double *anArray = new double[arraysize]; // 申一個 double 大小的空間 50個
```

* 陣列名稱 = 指標　anArray[2] ≡ *(anArray+2)

* 搬家。

```
double *oldArray = anArray // 50個 (舊資料)
anArray = new double[3 * arraysize]; // 150個 (新空間)
```

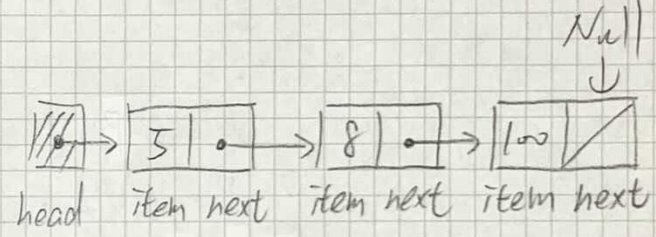
若 anArray 想要留有舊資料，需一行搬家

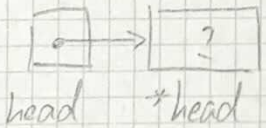
⇒

```
for(int i=0; i< arraysize; ++i)
    anArray[i] = oldArray[i];
delete[] oldArray; // 歸還所有房子
```

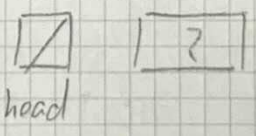
* 指標做鏈結串列

```
struct Node {
    int item;
    Node *next;
};
```






```
head = new Node
```




```
head = NULL
```

~ 被遺忘 (memory leak)
記得 delete