OS note 1092253 張名洋

2021-9-13. Basic of all fields in Computer Science

Data Structure

a way to store data (for easy use)

+

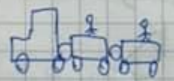
Algorithm 演算法

a finite set of instructions to complete a task

+

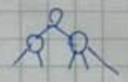
Coding

一些基本的資料結構

①串列 List  串在一起

②堆疊 Stack

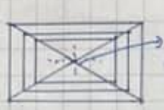
③佇列 Queue

④樹 Tree 

遞迴 (recursion) v.s. 迴圈 (iteration)

优点：簡单明瞭、精簡

缺点：效率可能不好



敘述：很像兩面鏡子的畫面，把問題越變越小

— Linear recursion — Binary recursion

Work in：

□ Factorial	階乘
□ Greatest Common Divisor	最大公因數
□ Search in Array	搜尋
□ Fibonacci series	費式數列
□ Combinatorial numbers	組合數
□ Towers of Hanoi	河內塔
□ fractal ……	碎形

· A binary search is recursive

→ Uses a divide and conquer strategy 分而擊之

divide：分割　conquer：征服

(HW)：反向印字：

→字串長度減1，字元慢慢變少

Box trace (箱型迴溯)

string：資料型態　size > 0：Base case　[illegible]

size-1：拿最後一個字元　1：一次一個

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

減成十　減成0，印第一個字元.

Ⅰ S = 'cat' S = 'ca' S = 'c' S = ' '

tac size = 3 size = 2 size = 1 size = 0

摘最後一个字元

Ⅱ S = 'cat' S = 'at' S = 't'

cat 摘第一个字元

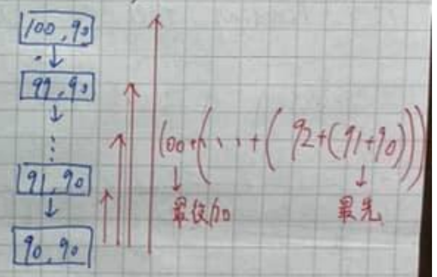
Ⅲ 倒过来.

tac

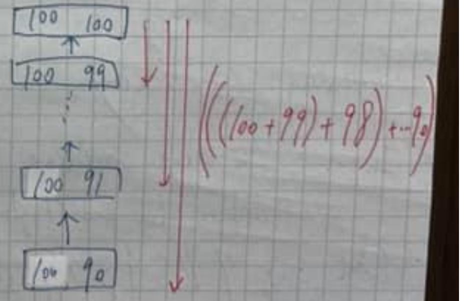
心得：Ⅰ 较快 其't'会先出来

Practice 1-1. Give two numbers a and b, where a > b, write a recursive function to compute the sum of all the integers from a to b, inclusively.

①.
```
int sum (int a, int b)
{ if (a > b)
    return sum(a-1, b) + a;
  else // a==b → 触底反弹
    return b ;
}
```



②.
```
int sum ( int a, int b)
{  if (b == a)
       return a;
   return sum(a, b+1) + b ;
}
```

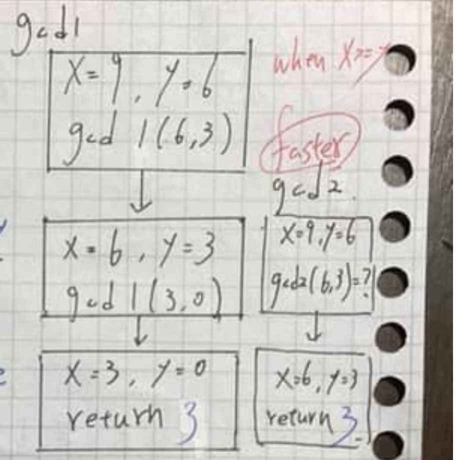


(HW) 找最大公因數

$$
\begin{aligned}
gcd1(x, y) &= x \quad \text{if } y = 0 \\
&= gcd1(x, y \bmod x) \quad \text{if } y > x \\
&= gcd1(y, x \bmod y) \quad \text{otherwise } x > y
\end{aligned}
$$

餘

$$
\begin{aligned}
gcd2(x, y) &= y \quad \text{if } x \bmod y = 0 \\
&= gcd2(y, x \bmod y) \quad \text{otherwise}
\end{aligned}
$$



gcd1

X = 9, y = 6
gcd 1(6,3)

X = 6, y = 3
gcd 1(3,0)

X = 3, y = 0
return 3

when X>=y

faster

gcd2

X=9, y=6
gcd2(6,3)=?

X=6, y=3
return 3

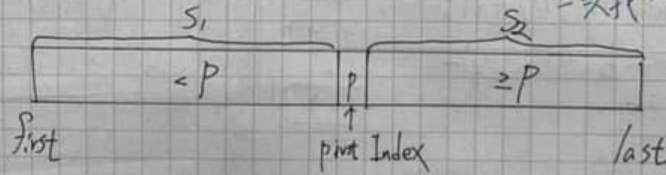
1-06、分而擊之的演算策略

Binary Search → 一分為二（縮小問題）

1-07. Finding the k^{th} Smallest Item in an Array

① Selecting a pivot item 樞紐

一次找一半!!



Revering an Array

Input : An array anArray and nonnegtive integer indices low and high.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

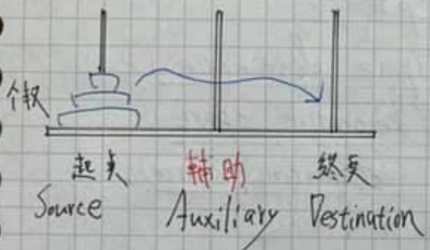
Out : | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

code:

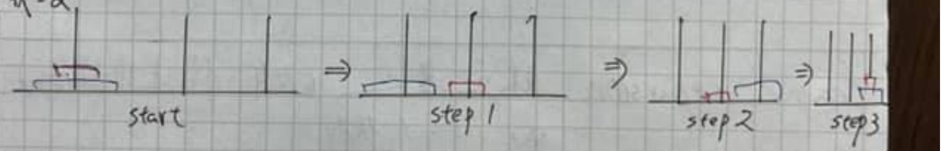
```
Algorithm ReverseArray ( anArray, low, high) {
    if (low < high)
```

```
      交換位置
    { Swap anArray[low] and anArray[high];
      ReverseArray(anArray, low+1, high-1);
      return;   ↘ 遞迴呼叫
    }
}
```

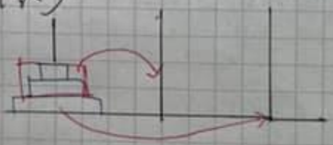
- Towers of Hanoi



個數=2



個數3



★ 把問題變小

$2^3-1=7$、

$2^4-1=15$、

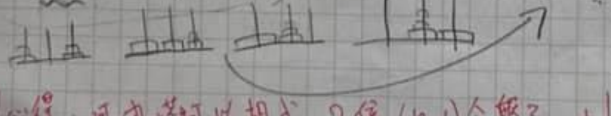
$\Rightarrow 2^{\boxed{n}}-1$ 次

盤數

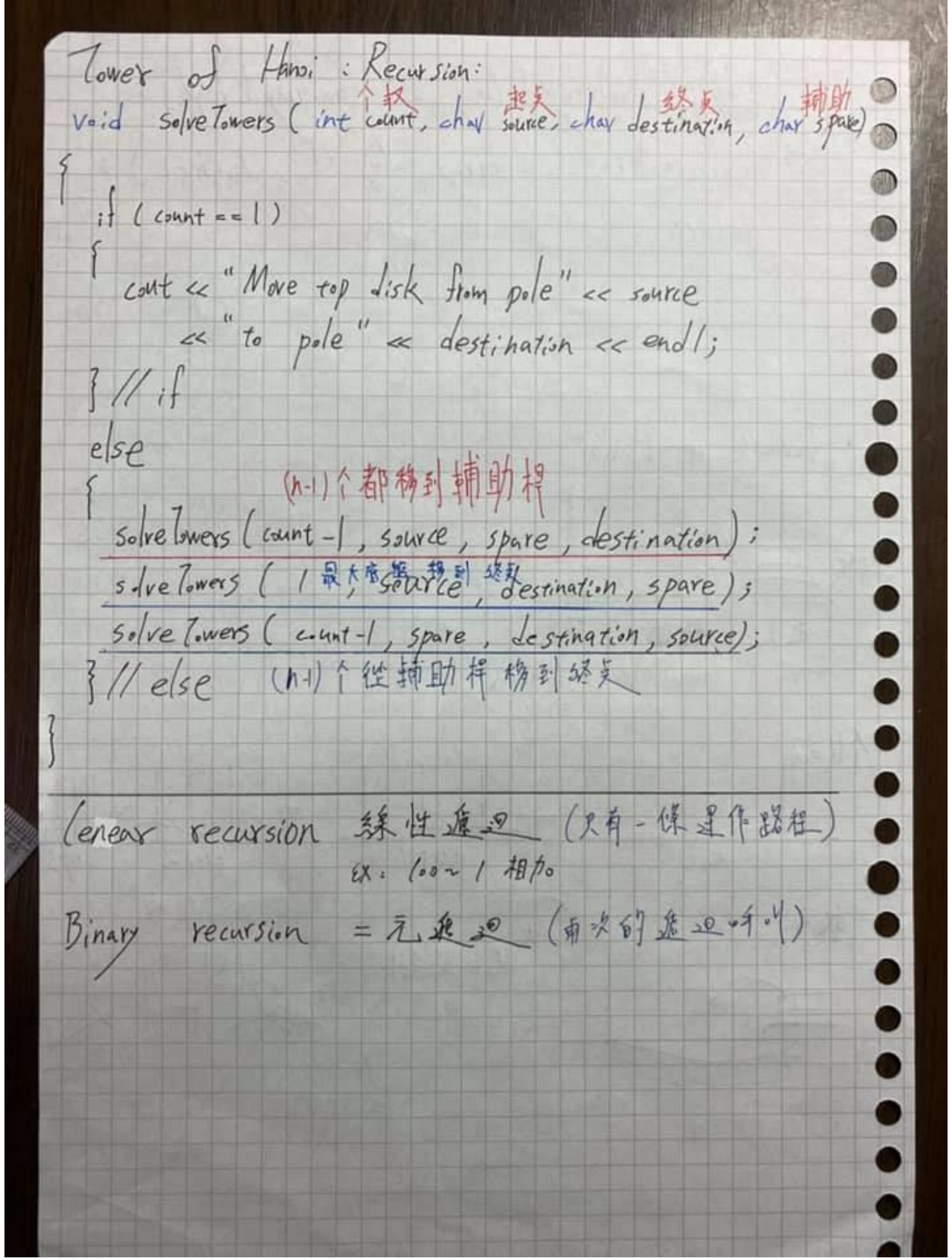
$30+1$ 31

63

$\underbrace{3} + 1 + 3 + 1 + \underbrace{7}$



♡心得：河內塔可以想成 2倍 $\underbrace{(n-1)\text{個盤子}}_{\text{搬}(n-1)\text{個}}$ $\underbrace{+1}_{\text{[illegible]}}$ 次



Tower of Hanoi : Recursion:

```
                          个數        起點          終點         輔助
void  solveTowers ( int count, char source, char destination, char spare)
{
   if ( count == 1 )
   {
      cout << "Move top disk from pole" << source
           << "to pole" << destination << endl;
   } // if
   else
   {               (n-1)个都移到輔助桿
      solveTowers (count-1, source, spare, destination);
                     最大底盤 移到 終點
      solveTowers ( 1, source, destination, spare);
      solveTowers ( count-1, spare, destination, source);
   } // else   (n-1)个從輔助桿移到終點
}
```

lenear recursion 線性遞迴 (只有一條運作路徑)

ex: 100~1 相加

Binary recursion 二元遞迴 (兩次的遞迴呼叫)

Multiplying Rabbits (Fibonacci)

□ Base cases

- rabbit(2), rabbit(1)

□ Recursive definition

呼叫次数以"指数"成長

$\text{rabbit}(n) = 1$, if n is 1 or 2

$= \text{rabbit}(n-1) + \text{rabbit}(n-2)$, if $n > 2$

此二元遞迴，沒效率!

// rabbit※(7) = rabbit※(6) + rabbit(5) + 1

呼叫次数　　叫自己

☆用空間換時間

※ Better Fibonacci

Use linear recursion instead

(3+2, 3) (2+1, 2) (1+1, 1) (1+0, 1) (1, 0)

5 → 4 → 3 → 2 → 1

```
Algorithm linearFibonacci(k)
    Input: A nonnegtive integer k
    Output: Pair of Fibonacci numbers $(F_k, F_{k-1})$
    if k=1 then
        return (k, 0)    // base cases : $k=1 \rightarrow (F_1, F_0)$
    else
        (i, j) = linearFibonacci(k-1)    // $(F_{k-1}, F_{k-2})$
        return (i+j, i)    // $(F_k = F_{k-1}+F_{k-2}, F_{k-1})$
```

1-15. 以遞迴求遊行隊伍排列數

Organizing a Parade

Problem: How many ways can you organize a parade of length "n"?

Subject: 樂隊不可緊跟樂隊

$$P_{(n)} = F_{(n)} + B_{(n)}$$

n个隊伍(3/2)的排列數 = (事件一)花車殿後 + (事件二)樂隊殿後

$F_{(n)} = P_{(n-1)}$ 只要處理前(n-1)个

$B_{(n)} = F_{(n-1)} = P_{(n-2)}$

只能在下一个放花車, 視同 "樂" "花" $\overline{\quad\quad}_{(n-2)}$ ……

$\Rightarrow P_{(n)} = P_{(n-1)} + P_{(n-2)}$, as Fibonacci serles.

base case

$P_{(1)} = 2$ 樂 or 花

$P_{(2)} = 3$ 樂花 or 花樂

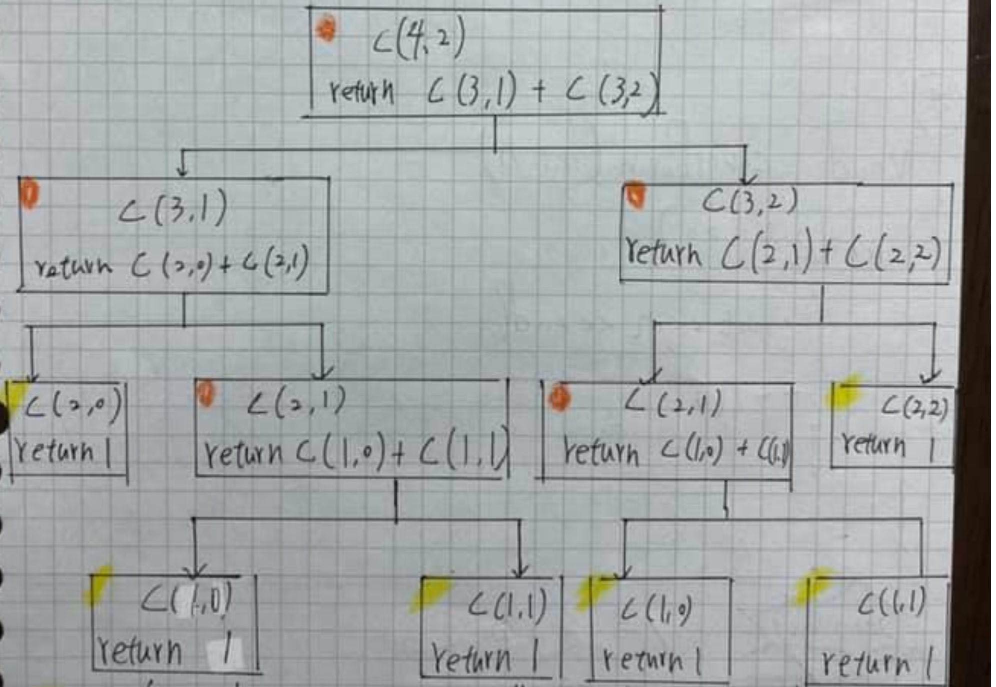
$P_{(3)} = 2+3$

$C^{k}_{n} = ?$ 如：8行星.

$C(n,k) = C(n-1,\underset{必選地球}{k-1}) + C(n-1,\underset{不選地球}{k})$

※ Base case 終止條件

- There is a group of everything $C(k,k)=1$
- There is a group of nothing $C(n,0)=1$
- Special $C(n,k)=0$, if $n>k$



● Leaf nodes : recursive calls to base cases 葉節點

● Internal nodes : recursive calls to non-base cases 內部節點

※ Leaf - Internal =1 [illegible] 數量差1

HI

可以把 tral Recursion 改成迴圈，效果更好
尾端遞迴　　　　　　　　　　　　　　　（能）

例：

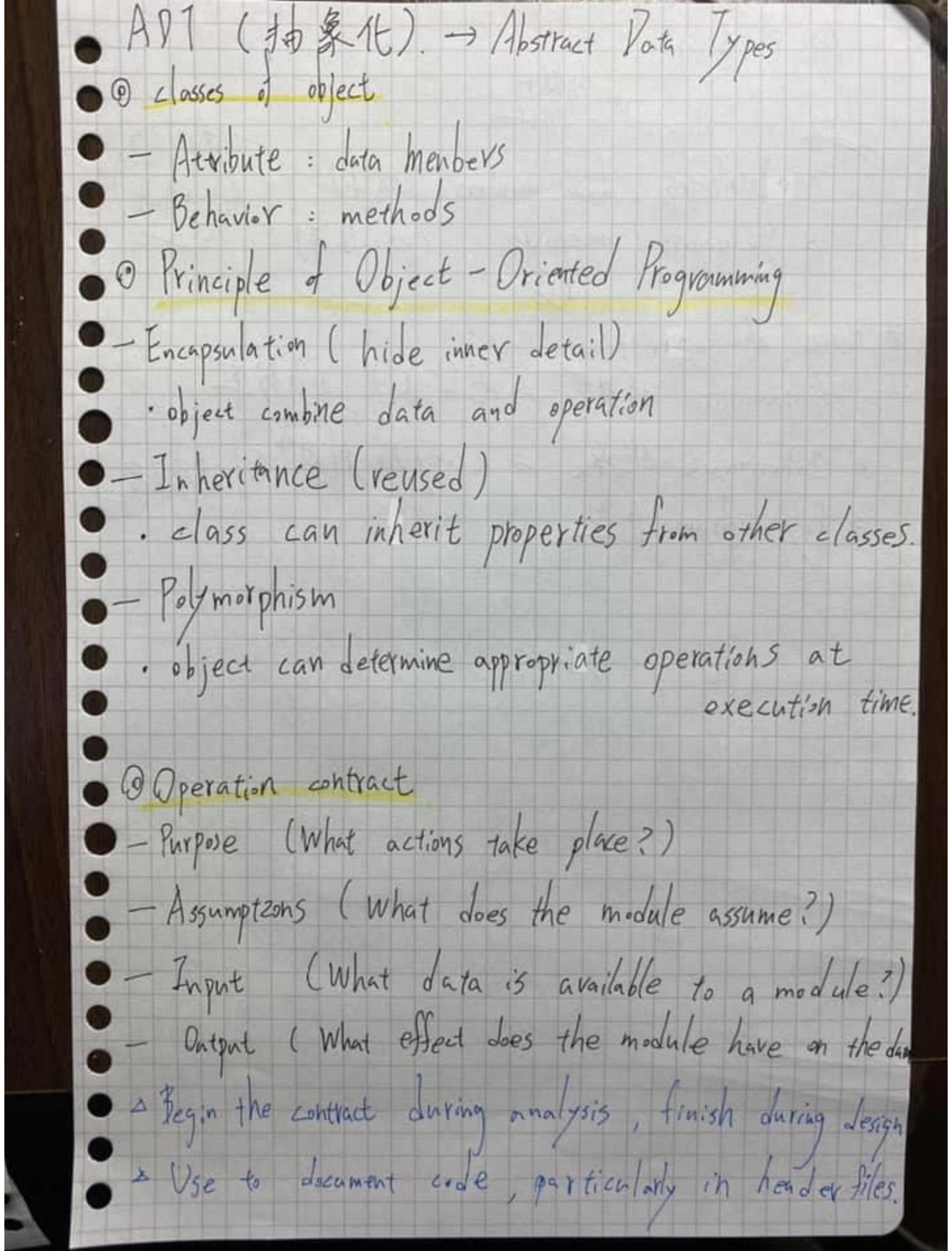
```
Void countDown (int n)
{ if (n>0)
    { cout << n << endl;
    ~~countDown~~(n-1);
    }
}
```

改：

```
Void countDown (int n)
{ while (n>0)
     { count << n << endl;
      --n;
     }
}
```

Summary

1. 遞迴定義
2. 問題簡化
3. 終止條件
4. 保證終止



ADT (抽象化). → Abstract Data Types

① classes of object

- Attribute : data members
- Behavior : methods

② Principle of Object-Oriented Programming

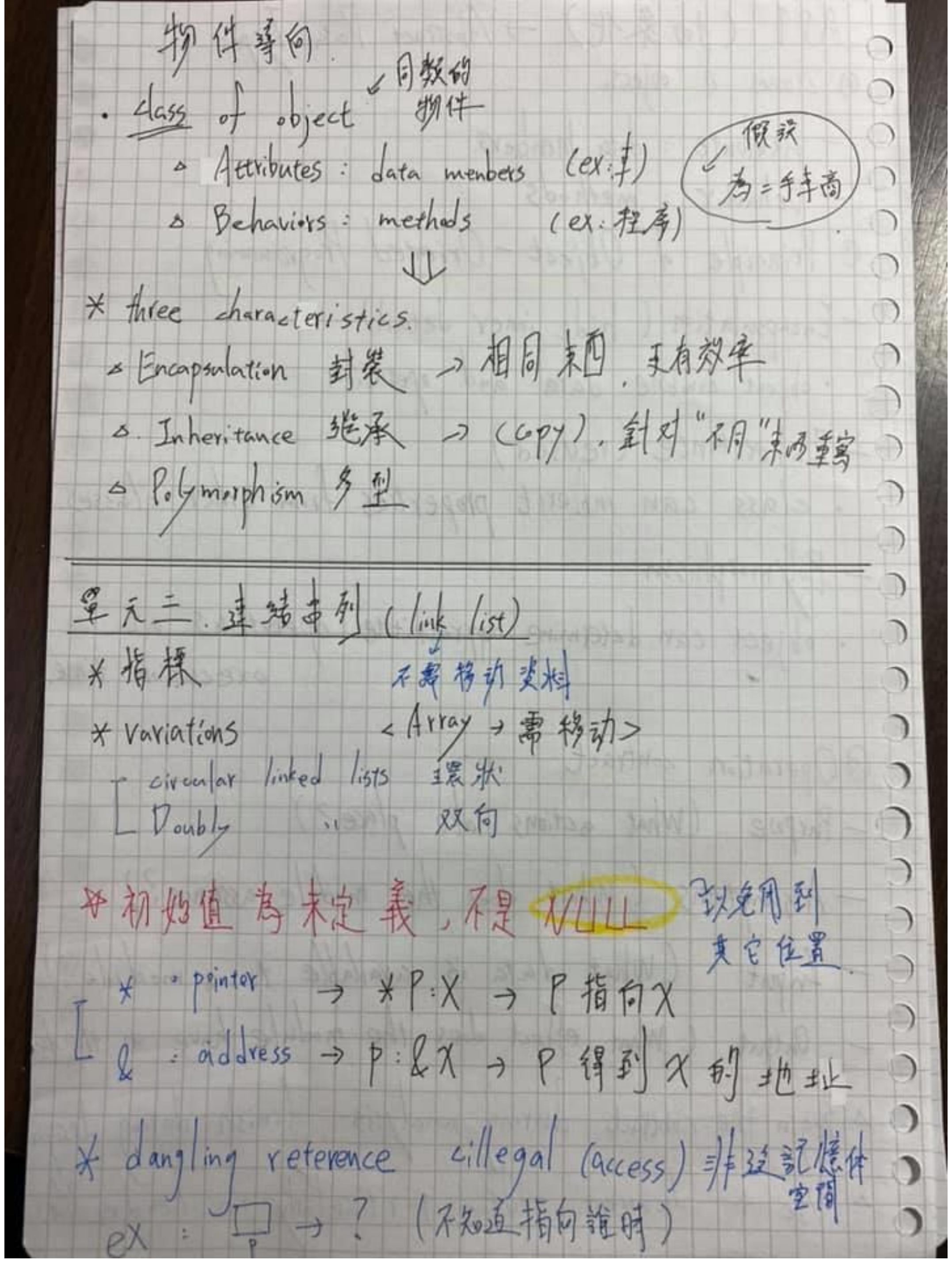
- Encapsulation (hide inner detail)
  - object combine data and operation
- Inheritance (reused)
  - class can inherit properties from other classes.
- Polymorphism
  - object can determine appropriate operations at execution time.

③ Operation contract

- Purpose (What actions take place?)
- Assumptzons (what does the module assume?)
- Input (What data is available to a module?)
- Output (What effect does the module have on the da

△ Begin the contract during analysis, finish during design

△ Use to document code, particularly in header files.



物件導向

- class of object ↙同類的物件
  - △ Attributes : data members (ex:手)
  - △ Behaviors : methods (ex:程序)

假設 為=手車商

⇓

* three characteristics.

- △ Encapsulation 封裝 → 相同東西，更有效率
- △. Inheritance 繼承 → (copy)，針对"不同"東西重寫
- △ Polymorphism 多型

單元二．連結串列 (link list)

不需移動資料

<Array → 需移動>

* 指標

* variations

- circular linked lists 環狀
- Doubly 〃 双向

* 初始值為未定義，不是 NULL 以免用到其它位置

- * : pointer → *P:X → P 指向X
- & : address → p:&X → P 得到X的地址

* dangling reference (illegal (access) 非法記憶體空間

ex : □ → ? (不知道指向誰時)

p

*动態(配置)陣列 (Array dynamically)

```
⇒ int Arraysize = 50
  double * anArray = new double [arraysize];
```

(註). anArray[2] ≐ *(anArray+2)

· dele [ ] anArray ⇒ 全部刪除
(釋放空間)