Recursion. 遞迴.

Date NO.

* 主要觀念：遞迴的問題有兩大要點."同樣的條件" & "越來越小的問題"
  滿足此二條件就可以用遞迴解題.

* Practice 1-1. (輸出 a=100, b=90, a+到b, 100+到90).

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

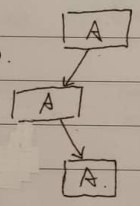
100, 90
↓
100, 91.
⋮
100, 100.

* 遞迴定義：
  sum(a+...b) = sum(a+...b+1)+b.
* 問題簡化： b => b+1.
* 終止條件： b, b+1, ... a => b==a
* 保證終止； natural numbers a > b.

* 沒效率的遞迴如果產生太大量可能會導致卡死.

* Linear Recursion.
  => 只有一邊要走.
  => 效率"可能"比.
  要走兩邊的高.

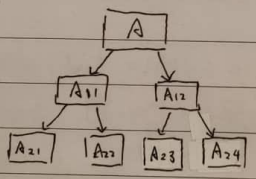
ex. =>.



像找第k小的. 要的資料一定在.
Pivot item 的左or右邊. 所以可以
極端. 只往遞迴的一邊走.

* Binary Recursion.
  => 有兩邊要走.

ex. =>



* 遞迴形式的題目
  => 河內塔，費式數列，找第k小的數值，尾端遞迴.

* Four questions about recursion.
  1. 遞迴定義：找出如何量化此遞迴.
  2. 問題簡化：看如何把問題縮小.
  3. 終止條件：設定 base case. 終止條件.
  4. 保證終止：確保所有情況都能停止，沒有例外.

DOW DEAN

Data Abstraction. 資料抽象化.

Date NO.

屬性. 運算.

* 主要觀念: Class 裡面會有 Attributes ⇒ data members, 以及 Behaviors ⇒ methods

三大性質: Encapsulation、Inheritance、Polymorphism.

封裝. 繼承. 多型.

* Operation Contracts.

⇒ 運算合約, 把這個程式要怎麼運行寫出來, 分為四點.

⇒ purpose、Assumptions、Input、Output.

* Abstract Data Type. (ADT). 描述 實做.

⇒ 模組化 (Modularity.): 分層管理, 有"Specifications"及"Implementation".

⇒ Specification: 定義 Operation 給其它程式使用. 要求解決問題.

⇒ Implementation: 從 Data Structure 去選適合的來解決問題. 要求效率.

* Class.

⇒ Constructors.: 和 Class 同名, 不用 return, 也不用 void. (建構).

⇒ 在記憶体內宣告一個空間用.

⇒ Destructors.: 清記憶体空間用. "~Class();" (解構).

⇒ 子類別: 在大 Class (父類別) 內再宣告一次 Class, 就是子類別.

⇒ 可以繼承父類別內的東西. (protected 內的). ex. class Int: public Ration

* Namespace.

⇒ 把自己常用的放在 namespace 內.

```
namespace smallNamespace {
    int count = 0;
    void abc();
}
using namespace smallNamespace;
count += 1;
abc();
```

* Exceptions.

⇒ try & catch.

```
try { ... throw (type) ... }
catch (type) { class.
    statements(s);
}
```

Linked List. 鏈結串列.

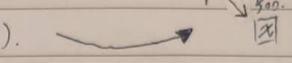
Date NO.

*主要觀念：像是一台火車，每個資料接到下個資料，然後尾端擋著.

* pointer.

=>. 指標 = 門牌. (int *p.(存門牌號碼)

p = &x. (&x => 房子x的門牌).



設為NULL前一定要delete.

- delete p; (歸還房子).
- p = NULL; (徹底遺忘門牌).

⇓

*如果把有存東西的指標在delete之前指向NULL，則稱為"Memory Lick."

*. Arrays.

=>. 動態配置陣列. int arraySize = 50;

double *anArray = new double[arraySize];

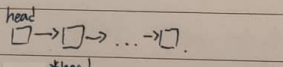
* Linked List.

=>. 結構. struct Node {

int item;

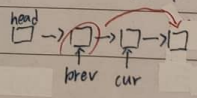
Node *next;

};



=> 新增. head = new Node; After head = NULL; =>

=> 刪除. ①. prev->next = cur->next.

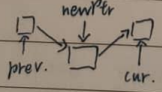


②. head = cur->next.

都接. cur->next = NULL;

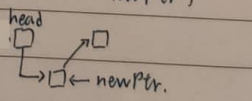
delete cur; & cur = NULL;

=> 插入. newPtr->next = cur.; prev->next = newPtr.;



newPtr->next = head;

head = newPtr;



* pointer 的順序很重要.