My Notes
Important Concepts worth keeping

單元一 recursive

Today: / /

*遞迴 recursive.

將問題分割成很多小問題，而小問題仍是本來的大問題，皆以相同的方式解決，所以用同一份程式碼可解決相似的題目，因此會更精簡、易解釋。

但不一定有效率

Ex1: Binary Search

① 一個一個找

② recursive → 策略: divide and conquer 分而擊之 (把問題分開，解決小問題)

base case

Sol: 先將資料由小到大排序

↓

從中間切一半，判斷尋找的值是大於or小於等於中間數

↓

若大於在右邊繼續找 or 若小於等於在左邊繼續找

↓

重複

*先找 base case

Punctuality: Showing esteem for others by doing the right things at the right time.

My Questions
Problems & Difficulties needing exploration

Ex2: Factorial 階層

$n! = n \times (n-1) \times (n-2) \cdots \times 1$ *base case: n=0, F(n)=1

$n>0$, $F(n) = n * F(n-1)$ 遞迴.

$n=0$, $F(n)=1$

Ex3: 反向印字串

* base case = 空字串

每一層輸出最後一個字元

* Recursive Solution = (1) 遞迴定義 (小問題)

(2) 問題簡化 (decreases the problem size)

(3) 終止條件 (set of base case)

(4) 保證終止 (reach a base case)

My Opinions
Thoughts, inspirations, and suggestions

一定要確定"中止"

*不是所有 recursive 都有效率

↳ Ex: 費式數列 (1, 1, 2, 3, 5, 8, ...) ⇒ 會重複呼叫

1st	2nd	3rd	4th	5th
1	1	2	3	5
base case		[1] [2]	[2] [3]	[3] [4]
一次數	1	1	3	5	9
呼叫					

呼叫 recursive 次數

以"指數"成長 ⇒ $n_k \geq 2^{\frac{k}{2}}$

Sol: ①可以用空間換時間 (把算過的放陣列中)

②使用產生式

守時：在對的時間，做對的事，來表明對別人的尊重。 -《培基》

3

My Notes
Important Concepts worth keeping

Today: / /

Ex4: n中選k

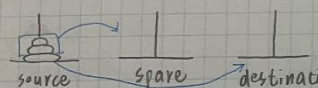
base case: $C(n,k) = 1$ if $k=0$ (什麼都不選)

$= 1$ if $k=n$ (全選)

$= 0$ if $k>n$ (選超過n個)

⇒ $C(n-1,k-1) + C(n-1,k)$, if $0<k<n$

Ex5: Towers of Hanoi




```
if(count==1) 結束
else{
    Towers(count-1, source, spare, destination); 除最下面移
    Towers(1, source, destination, spare); 最下面
    Towers(count-1, spare, destination, source);
}
```

開始 結束 重複

We are convinced that love is the principal guiding force in education. We, teachers and students alike, pursue mutual growth through instruction by both words and deeds, in a spirit of love and respect for one another.

-《CYCU Education Philosophy》

4

My Questions
Problems & Difficulties needing exploration

單元二 物件導向

描述ADT運算在前，之後進行ADT實作

Data Abstraction

* Object-oriented (物件導向)

↳ build class of objects (called instance)

* class:

- Attributes (屬性) → Typically data → called data members (adj./n.)
- Behaviors (運算) → Typically operate on the data → called methods or member function (v.)

* 三個物件導向的方法

- ~~Encapsulation (封裝)~~ → Objects combine data and operations, → hides inner details
- ~~Inheritance (繼承)~~ → 其他 class 可以繼承 → reused (超連結)
- ~~Polymorphism (多型)~~ → 可以決定適合的操作 (擴充)

Abstraction Data types (ADT): motive

- Modularity 模組化：有系統，易改易讀
  - ↳ Functional abstraction → specification 描述
    - implementation 實作 (隱藏)
    - (資訊隱藏)

我們確信『愛』是教育的主導力量，願以身教言教的方式，互愛互敬的態度，師生共同追求成長。 -《中原大學教育理念》

5

My Notes
Important Concepts worth keeping

Today: / /

Better solution:

① Cohesion (高內聚): 盡可能讓每個模組只做一件事

② Coupling (低耦合): 讓每一函式 or 模組關係不大(獨立)

ADT: concepts

- The isolation of modules is not total

ADT: goals

- Data abstraction

↳ 對一群資料做何事, 講明白; 怎麼做

結論:

ADT < A collection of data
A set of operations on that data.

[specifications (描述) = what the ADT operations do?
[Implementation (實作) = Includes choosing a particular data structure.

ADT operations isolate a data structure.

My Questions
Problems & Difficulties needing exploration

→ 隱藏實作細節

✓ class C++ (Encapsulation)

* A class contains data member and methods.
* By default, all members in a class are private.

[描述 ⇒ class.h → (header file)
[實作 ⇒ class.cpp

* Constructors (建構): 創建, 初始化, 名字跟 class 同
* Destructors (解構): 變數使用完消失 (釋放記憶體) (程式結束後)
Each class has one destructor

My Opinions
Thoughts, inspirations, and suggestions

✓ Class C++ (Inheritance)

* A derived class (子類) or subclass can invoke public methods of the base case (父類) or superclass
↳ 可衍伸

My Notes
Important Concepts worth keeping

(同一個 class 中)

Today: / /

* Overloading (多載) = 可宣告相同名稱不同參數的函式
* Overriding (覆載) = class間有繼承關係, 允許同名同參數的函式

[public = any class instances
[protected = subclass (子類) instances (具繼承關係的class可用)
[private = only class instances

✓ C++ Namespaces (管理宣告)

* the "using" declaration allows the names of elements to be used directly. (使用命名空間)

自訂命名空間: namespace Name { ... }
使用命名空間: using namespace Name;

✓ C++ Exceptions (例外狀況)

* Uses a try block and catch blocks
設定保護範圍 捕捉例外狀況

```
try { ... throw (type); ... }
catch (type1) {
  statement(s);
}
catch ( ... ) {
  statement(s);
}
```

↳ 記得釋放記憶體

可利用順序, 看type來辨認
throw → catch → 跳脫/離開

ONCE, all the villagers decided to pray for rain. On the day of prayer all the people gathered, but only one boy came with an umbrella.
That's FAITH!

My Notes
Important Concepts worth keeping

單元3 Linked lists

Today: / /

Pointer

A Pointer contains the location, or address in memory, of a memory cell.

int *p; (not NULL) 靜態宣告=直接配給 (p→位置, *p→值)

p=&x;

p= new int; 動態配置 Dynamic allocation

delete p; 記得釋放 memory

p=NULL; 避免讓其他函式用到錯的位置

p=&x (先)
*p=6 (後)

※注意 memory leak ※ (記憶體浪費,被遺忘的空間)
設定NULL前要先delete ※

動態(配置)陣列

int array =50;

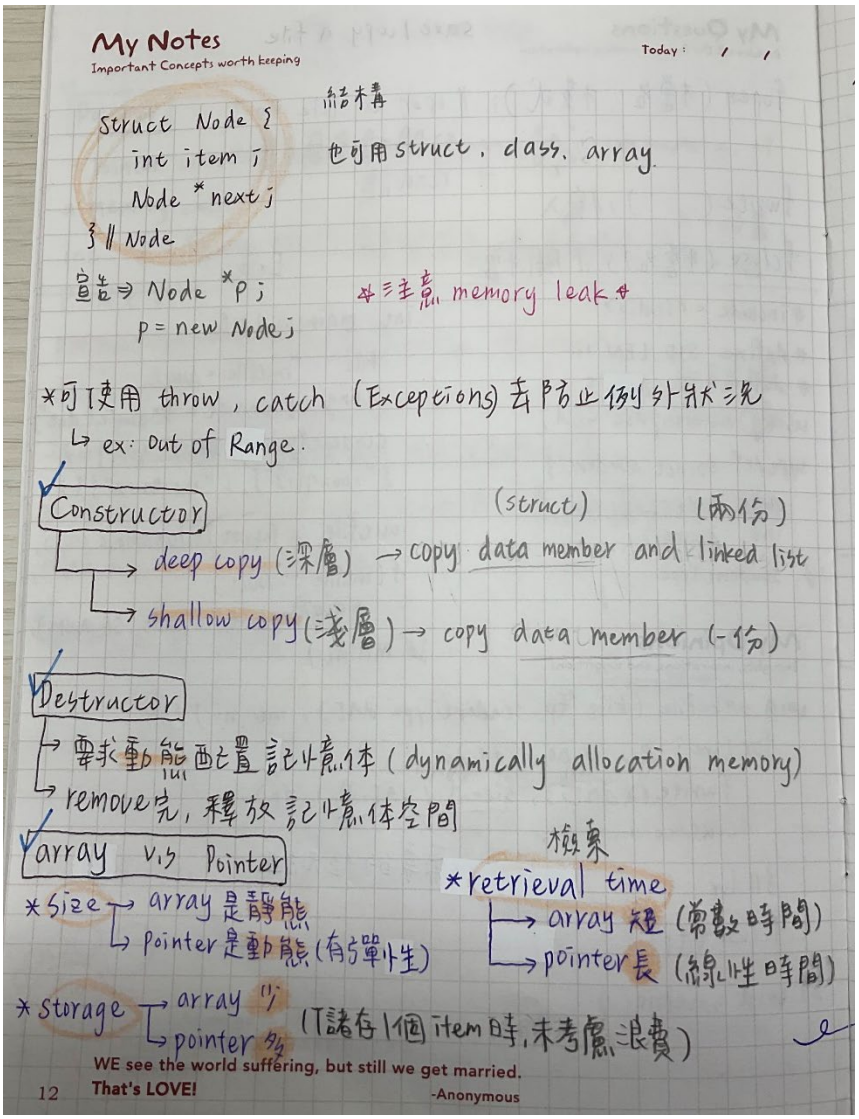
double *anArray = new double [array]; = 3*array 增加空間

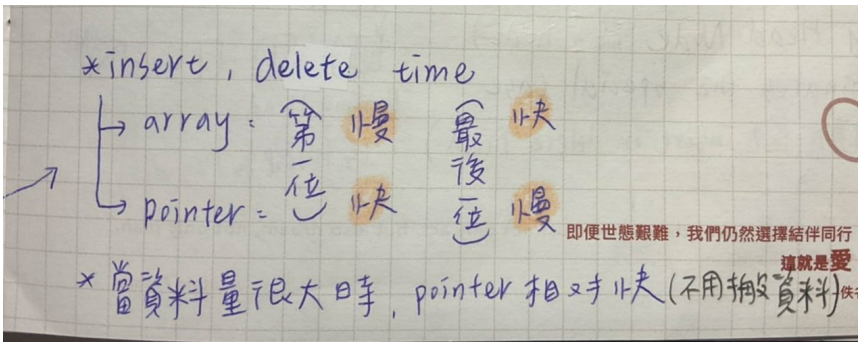
int anArray[2] (=) *(anArray+2) → 相當於 位置往後2格

動態陣列缺失:舊搬新要一個一個搬

刪一群 ⇒ delete [ ] oldarray;
(一群)

EVERY night we go to bed, without any assurance of being alive the next morning, but still we set the alarms.


My Notes
Important Concepts worth keeping
Today: / /
結構體
Struct Node {
int item;
Node *next;
} // Node
也可用 struct, class, array
宣告 ⇒ Node *p;
p = new Node;
注意 memory leak
*可使用 throw, catch (Exceptions) 去防止例外狀況
↳ ex: out of Range.
Constructor
(struct) (兩份)
deep copy (深層) → copy data member and linked list
shallow copy (淺層) → copy data member (一份)
Destructor
↳ 要求動態配置記憶体 (dynamically allocation memory)
↳ remove完, 釋放記憶体空間
array v.s Pointer
*size → array 是靜態
↳ pointer 是動態 (有5單小生)
*storage → array 少
↳ pointer 多
(IT諸存1個item時, 未考慮浪費)
檢索
*retrieval time
↳ array 短 (常數時間)
↳ pointer 長 (線性時間)
WE see the world suffering, but still we get married.
That's LOVE!
-Anonymous
12


*insert, delete time
↳ array: 第一位 慢 最後一位 快
↳ pointer: 第一位 快 最後一位 慢
即便世態艱難，我們仍然選擇結伴同行
連就是愛
*當資料量很大時, pointer 相對快 (不用搬資料)

My Notes
Important Concepts worth keeping

存檔

Today: / /

saving and restoring a Linked list

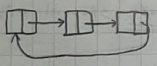
* Write only data to a file, not pointer
* Recreate the list (重建，循序加入尾端)

```
ofstream ofs(filename);  // 宣告輸出檔案 (檔名)
ofs << item << endl;     // 寫入檔案 (與 cout 用法相同)
ofs.close();             // 關輸出檔案
ifstream ifs(filename);  // 宣告輸入檔案 (檔名)
ifs >> item;             // 讀檔 (與 cin 用法相同)
ifs.close();             // 關輸入檔案
```

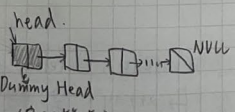
* recursive view of a sorted linked list
  ↳ 每次遞迴後印第一個 → 刪除第一個

Variations

* Circular Linked list (環狀):
  - Last node points to the first node
  - Every node has a successor



* Dummy Head Node (設空 node)
  - Eliminates the special case (不用考慮是否 insert or delete head)



To accomplish great things, we must not only act, but also dream, not only plan, but also believe.

-Anatole France

My Questions
Problems & Difficulties needing exploration

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

* Double linked list (雙向):
  - Each node points to both its predecessor and its successor
  - insert / delete 要注意有多一個指標要處理
  - 通常結合 Dummy Head Node 和 Circular Linked list.
  - 使用空間最多

單元十 Grammars

判斷讀入是否滿足 <digit>

文法定義 ← 規則

<number> = <digit> | <digit> <number> (or) Ex. 1235 true

<digit> = 0|1|2|3|4|5|6|7|8|9 12+5 false.

(1個以)

<number> = <digit> + <number> | <digit> 判斷是否是加法

↳ 只要判斷個位數加法 Ex: 1+3 true. 12+3 false 1+x

遞迴

base case

My Opinions
Thoughts, inspirations, and suggestions

* 兩位數以上的加法

<add> = <num> + <add> | <num> ⇒ 加法是否合法.

<num> = <num> <digit> | <digit> ⇒ num 為多位數 or

<digit> = 0|1|2|3|4|5|6|7|8|9 一位數.

⇒ 用遞迴處理.

(下頁)

grammar → 想法描述
code → 想法執行

密碼 cipher

為成就大事，我們需要的不只是行動，還要夢想，不只是計畫，還要信心。

-阿納托爾·法郎士

My Questions
Problems & Difficulties needing exploration

Expression 運算式

* 中序式 (Infix) = 一個運算子 (operator) 兩旁是運算元 (operands) 1+2
* 前序式 (Prefix): 運算子放運算元前面 +12
* 後序式 (Postfix): 運算子放運算元後面 12+

中序 Grammars: (都加括號的前題)

<infix> = <identifier> | (<infix> <operator> <infix>)

<operator> = + | - | * | /

<identifier> = 1|2|3|...|9 → 可加多位數.

前序、後序 → 找相應的括號專模.

↳ 優點 = No 優先權, No 括號, No 結合律.

* 一個前序式後面再接上非空字串一定不是前序式 *

My Opinions
Thoughts, inspirations, and suggestions