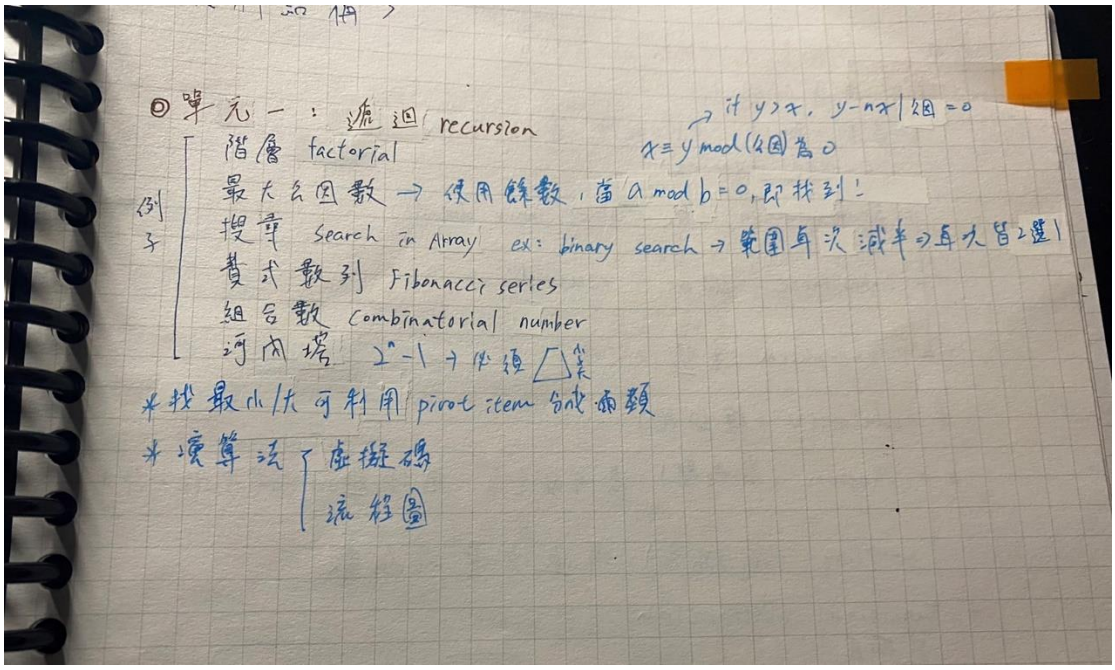
資結筆記 單元 1~3

10826223 資工三甲 廖悅婷

- 筆記：



⑧ 單元二：抽象化 (Data abstraction)

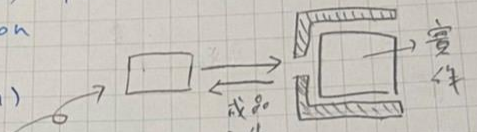
* 物件導向觀念
  ex: 資料 = 物件，事主 = 物件，約為 = 方法
  (called instances)

- Encapsulation 封裝：hides inner details
- Inheritance 繼承：reused
- Polymorphism 多型：

* Modularity (模組化)

* - Cohesion (高內聚)：盡可能多每一個模組 or 函式只做一件事
  - Coupling (低耦合)：傳遞少量的參數

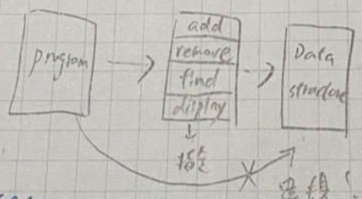
* Functional abstraction
  - 描述
  - 實作 (implementation)



* Data abstraction：
  不用了解怎麼達成，只要說要做什麼即可

* Abstract Data Type (ADT)
  不考慮實作細節，把概念表達清楚即可。

ex:



* C++ classes
  - 封裝 (encapsulated)
    data members & methods
    私密 or 公開
    header file ex: className.h
    實作 ex: .cpp
  - constructors：通常和class名字一樣，做預設用，
    Destructors：跑完後會自動刪除 instance of object

ex:
```
// file sphere.h
class Sphere {
  public:
    Sphere();
    ⋮
  private:
    double theR();
    ⋮
};  // end class
```

* class 呼叫 class 要 include !
ex: #include "sphere.h"
→ 使用時: sphere s = sphere()
{
}// end .

* class Colored Sphere : public Sphere → derived class (子)
子類別 (小) 父 (大) super class (父)
(subclass) base class (父)
→ is-a 關係 (private 不可繼承)

* overriding 覆載 → 修改繼承的部分
overloading 多載 → 參數列不同

* c++ namespace
// 範圍解析. using
ex: namespace Name
{ int a=0;
}
using namespace Name;

* c++ exception (例外)
#include <stdexcept>
ex: ① try → try block 設定保護範圍
{
statements();
}
② catch(ExceptionClass, identifier) → 捕捉例外狀況
{ statments();
}
③ catch(...) { // 跳脫, 類似 switch.
statements();
}

ex: void myMethod(int x) throw(MyException)
↳ 例外規格
{
if(...)
throw MyException("...")
}

第九章 鏈結串列 (Link list)

* 指標 (pointer) → 不需移動資料 (Array → 需要移動)
* variations
  - circular linked lists 環狀
  - Doubly linked lists 雙向

pointer

ex: → → → //

可以 add 或 dele

刪除:
delete p;
申請新 operator
→ p = new int;

指標可當作門牌號碼 → *, ex: int *p;

* 初始值為未定義 → safeguard
  不是 NULL!
  在刪除後，再設成 NULL → 以免用到其他位置

bad_alloc
↓
error，記憶體配置失敗

pointer p

- * : pointer → *p = 7 → p 指向 7
- & : address → p = &7 → p 得到 7 的地址

*p = 6 → p → 6 (7 or *p)

p = new int ⇒ p → ? (*p) 6 (7)

delete p;
↓
p = NULL;

p → 原來連接的東西即刪除

如果不先 dele 直接把 p 改為 NULL，則之前連接的東西不會被刪掉，也很難找回 → 浪費空間! (memory leak)

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

ex: i → ? (不知指向誰時)

p

* 動態 (配置) 陣列 (Array dynamically)

⇒ int arraySize = 50;
double *anArray = new double [arraySize];

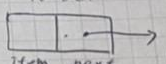
P.S. anArray[2] ≡ *(anArray+2) → 差不多的意思

⇒ dele [] anArray; → 全部刪除

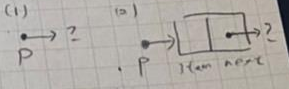
* file 存/讀檔 ⇒ 3-05
* pointer 連接.

ex.

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




```
(1) Node *p;
(2) p = new Node;
```



[shallow copy (淺): 沒再複製, 直接做更動.
[deep copy (深): 複製一份, 更動複製的那份.

* include <cstddef> // for NULL
  <new> // for bad_alloc
* 可用 try & catch 以防 bad_alloc

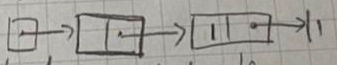
```
try
{ ...
}
catch (bad_alloc)
{ ...
}
```

* Array 比 link list 較省空間
* 存資料, 不存 pointer (restoring)
  ↳ 因電腦關掉, 記憶体位置會 reset

[ofstream 輸出 << head->item 寫入到 ofstream 的檔
[ifstream 輸入 >> 到 head->item

* Dummy Head Node: 節點內沒有存放東西
  ↳ 最後會刪掉

ex:



↘ 在刪東西比較方便插入