C++ 程式設計

丁 培 毅

97/02 – 97/06

http://squall.cs.ntou.edu.tw/cpp/

http://sirius.cs.ntou.edu.tw/cppBB/

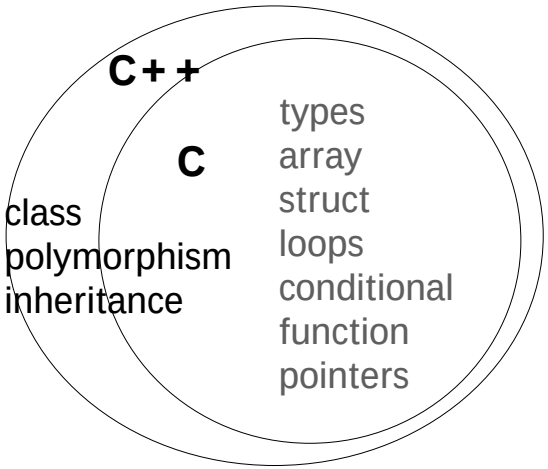
物件導向應用軟體開發過程

物件導向分析
Object-Oriented Analysis, OOA

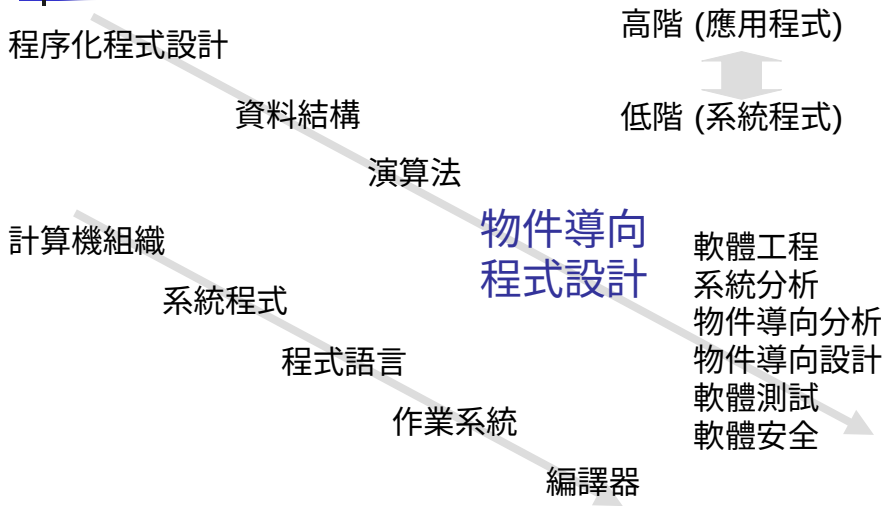
物件導向設計
Object-Oriented Design, OOD

物件導向程式設計
Object-Oriented Programming, OOP

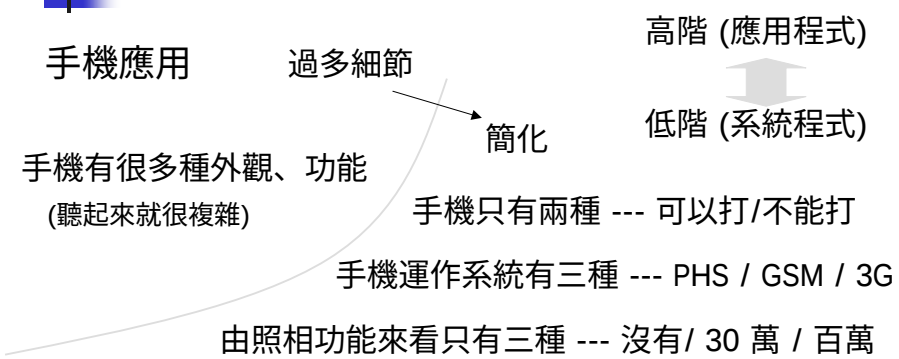
學習 OOP 的媒介



學習軟體設計的過程



抽象化 (簡化) 過程



抽象化 (簡化) 過程 (cont'd)

- 抽象化 (Abstraction):

  去掉不在意的細節, 專注於所在意的性質

- 在 OOP 中以

  介面 (interface)

  來實作抽象化的過程

OOP 重用"軟體設計"

Reuse

- 縮短軟體的生產過程
- 降低軟體更新功能時的阻力、縮短修改的流程
- 減少不必要的錯誤

- 封裝 (Encapsulation)

  確保實作介面功能的正確性, 降低軟體的成本
- 繼承 (Inheritance)

  重用軟體的架構 (程式碼與介面)
- 多型 (Polymorphism)

  基於抽象介面撰寫程式, 使用不同實體物件的機制

教科書

- Thinking in C++, Bruce Eckel, 2nd Ed., http://www.bruceeckel.com/
- 課程網頁 http://squall.cs.ntou.edu.tw/cpp/
  - 參考書, 參考資料
  - 課程內容摘要, 投影片
  - 作業
  - 實習
  - 討論群組
  - 過去課程網頁

軟體環境

- Compiler: VC 6.0
- 開發環境:
  - IDE: Visual Studio 6.0
  - Command Line: VC 6.0

如何上課

- 請預習
  - 課本
  - 中文參考書
  - 投影片, 講義
- 上課
  - 請隨時準備問題, 這和你未來是否留在資訊業界有密切關係
- 請複習
  - 定期單元小考
  - 不定期測試

我們每個人都生存在浩瀚的知識海洋中

在這個迅息萬變的環境中學習到的知識因人而異

學校與課堂中希望能夠幫助你尋找一條你自己的路

你, 同學, 助教, 和老師應該要充分合作才能幫助你收穫你想要的

『學問』…要學也要問

- 問問題是有方法的
  - 會問問題的人是能夠掌握自己目標的人
  - 是隨時動腦的人
  - 是能夠 (願意) 和同僚互動的人 (心態常比能力還重要)
- 問得好, 你很容易得到你想要的, 解決你的困惑, 沈澱自己的想法, 被問的人甚至由你的問題中獲益
  - 大部分主管/老闆其實都不知道底下的員工整天在做什麼, 可是常常幾個問題就可以掌握 80%
- 問得不好, 隨便問, 太籠統, 吹毛求疵, 被問的人不知從何答起, 他的回答也幾乎對你沒有作用

發問

- 問問題需要練習嗎?
  - 當然需要, 很多同學遇到需要問問題時腦子裡一片空白
  - 在學校裡練習問問題你沒有損失, 只有獲得, 老師也沒有損失, 同學也會有其它獲得
  - 有些同學會用影響課程進度作為不問問題的藉口, 那要看你問什麼問題, 不問看看你怎麼知道? 沒有常常看到同學在問問題, 你怎麼能學到怎樣問問題才是有用的?
  - 怕老師答不出來 (真好心, 不過老師也有很多招數的啊)
  - Trick: 有的時候問問題可以取代答案…
- 要問得好, 需要了解問題的答案嗎?
  - 雞生蛋蛋生雞
  - 不需要, 也不可能

發問

- 怎麼會透過問問題來學習呢?
  - 要別人直接幫你思考, 這樣會不會太偷懶了啊?
  - 很多時候你在組織問題的時候就是強迫你自己在壓力下思考, 分析所有手邊的資料, 讓你的腦子裡一些不常用的部份一起發揮功能, 常常會有意想不到的結果
  - 這是增加探討問題深度的主要方法
- 方法
  - 需要從不同的面相去觀察一個主題
  - 可以由如何應用, 用途, 功能性來探討; 可以由動機來探討; 可以由本身的推理邏輯來探討; 可以由其他配合的環境來探討; 可以探討效率的問題; 可以由其他相類似的方法來橫向的探討…

發問

- 期勉

  當我們這一班能夠不斷地討論問題的時候
  - 大家可以觀察到老師呈現知識的步驟, 可以觀察到同學在這個環境中如何思考, 解決問題的步驟
    -- 否則用一片教學 VCD 可能可以達到更大的效果
  - 大家才能夠真正運用到上課的時間深入學習, 所獲得 **深度的知識** 與 **探索知識的方法** 常常可以幫助你探討其他沒有在課堂裡談到的知識 (不要嫌上課得到的東西太少, 老師教得太少…幫你釣再多的魚你沒辦法自己如法炮製終究是一種浪費… 見山不是山見水不是水也不過是好高騖遠, 終究無法體會)

程式學不會

沒興趣

我覺得熱門音樂都是一樣的 --- 都很吵

我覺得古典音樂都是一樣的 --- 都很沈悶

我覺得籃球沒什麼好看的 --- 一個球十個人搶來搶去

我覺得物理、數學很無聊 --- 式子一堆太繁瑣, 我該不會用到吧

那麼程式設計呢?? 怎麼寫都有錯誤…

資訊系畢業不見得一定要寫程式吧?!

學不會?

- 嚴格來說, 學不會是一種藉口
  - 沒有付出足夠多的代價, 去把東西弄清楚
  - 你花的時間有比玩 Game 的時間多嗎? 比上網哈拉的時間多嗎? (和你有興趣的東西比較看看吧)
- 在大學裡這麼多科目的訓練, 希望你收穫到的是 "怎麼學會一種知識的方法", "建立怎樣學會一門知識的信心", 如果你真的對某樣東西沒有興趣時再換就是了, 如果從來沒有付出代價學會過什麼東西,一直在等待一個你可以不用付出太多就可以獲得很多的行業…

資訊業的特性

- 當然有這樣的行業, 不過通常也適合大部分的人, 也就是說競爭會很激烈, 手段會很殘酷, 場面會很血腥
- 表面上看起來資訊業就是這樣的一個行業, 不管你是什麼系畢業的, 經過三個月、六個月的訓練你就是專家了…(不相信你去外面軟體公司或是公司的軟體系統部門看看)
- 看起來門檻很低,人員的取代性極高…
- 既然如此, 為什麼各位要那麼辛苦地學習呢?

資訊業的特性 (cont'd)

- 你被人家取代的機會高嗎?
  - 你對系統的瞭解深入嗎? (你能夠修改作業系統核心嗎? 你能夠設計驅動程式嗎?)
  - 你能夠撰寫網路應用程式嗎? 圖形介面?
  - 你能夠設計大型的物件化應用程式嗎?
  - 你能夠設計複雜的知識存取、檢索核心嗎?
  - 你能夠設計高階的 3D 圖形介面嗎?
  - 你能夠設計智慧型的應用嗎 (語言轉譯、人工智慧、影像、視訊、語音輸出入…)?
  - 你能夠設計安全的軟體和通訊系統嗎?

資訊業的特性 (cont'd)

- 簡單地說…你希望建立一道知識與技術的防火牆…你希望透過幾年的苦讀突破瓶頸, 建立別人無法在短時間裡追上的屏障 …
- 沒有一種知識有絕對困難的門檻…就算有,也不見得每一個人都會遇見相同的瓶頸…
- 回應先前的說法, 這個門檻其實是你自己學習與應用的瓶頸…是那個一遇到困難立刻退縮的本能反應
- 很多同學到了三、四年級看到系上選修課程, 只想要找簡單的, 營養的, 不用花太多時間的課程選修
- 於是就加入了容易被取代的一群, 甚至還沒有畢業就已經開始尋求第二專長了, 資訊業不是很缺人嗎?

條條大陸通羅馬

沒有一條路、一個固定的模式適合所有的人

你自己找到的方法 才是對你最有用的,

雖然該滿足的**標準**可能由不得你

不過切記 有準備的人才能掌握機會

準備什麼?

- 準備一種接受挑戰的心態
- 準備一種系統地分析、解決問題的方法
- 準備一種尋找知識、接收知識、轉化知識為工作能量的成功模式
- 準備一種隨時面對環境變化的信心
- 準備一種逐步驗收成果的工程信念

抽象嗎?

就在當下

- 有的人永遠看前面, 看哪一條路所需要付出的和所能夠獲得的比例最高 (如果是買東西的話就是 cp 值囉)
- 在萬象的世界裡, “尋找” 所需要花費的時間是很冗長的, 可以保證你一定找不到“最好的” 一條路徑, 做出“最好的” 選擇
- 於是你必須要下賭注, 做選擇, 一旦下了決定就必須全力實現, 在有限的時間裡掌握所接觸的一切, 就是對你的未來做好最萬全的準備

怎麼調整你自己的學習方法?

- 很多接觸軟體開發設計的人都曾經萌生退意…(你呢?)
  - 開發工具變化好快
  - 系統好複雜, 好大
  - 設計的選擇性好多
  - 使用者的要求無理
  - 掌握不到軟體設計的精髓, 覺得好像不知道該要求些什麼, 有點不得其門而入

怎麼調整你自己的學習方法?

- 老師、同學、甚至網友都是你的鏡子
- 覺得沒有效率或是沒有成效的話, 自己思考一下, 整理一下, 和別人討論自己的心得
- 不要期待別人給你什麼實質的回饋,

  自己整理問題和心得時一定會有新的體會
- 凡事總要自己提出自己的看法
- 再由同學、課本、課堂、實習中驗證

其它工程系所 vs. 資訊工程

- 機械、造船、土木
  - 將來做出來的東西很具體
  - 基礎學習過程比較辛苦, 不知道什麼時候可以真的踏進生產線
  - 模型
- 資訊
  - 成品有點抽象
  - 學習的過程中不斷地可以作出成果出來
  - 模型?

軟體的特性與要求

- 軟體之所謂軟…因為沒有 “硬性” 不可變、不可挑戰的規則
  - 好處: 彈性很大, 山不轉路轉, 沒有標準答案, 正常運作就好…
  - 壞處: 很多小問題合在一起不斷放大, 到處藏污納垢, 沒有標準答案, 不知道到底對了沒有
- 解決方法
  - Coding styles, test-driven
  - 元件化
  - 模型化 (資料結構, 演算法, 物件化, 軟體模式)

對於大家的期許

- 過去十年在海大資訊系, 看到很多同學到了升大三的時候開始擔心自己的未來, 開始尋找必勝的捷徑, 於是看到有升研究所的補習班, 什麼都可以補, 既然看不到很明確的未來, 為何不去補習, 吸收一些考試的技巧, 藉由將來台大、清大、交大研究所的光環, 也許能夠少奮鬥幾年
- 考試是“結果論”的, 會考試當然不代表能夠解決實際的問題, 反倒是把時間放在準備考試上, 你一定無法兼顧而有實質的損失

對於大家的期許 (cont’d)

- 你已經經歷了十年考試制度的摧殘, 你也已經能夠抵抗這個壓力了, 還要再投入這個錯誤制度的循環?
- 你注意分析一個現象, 為什麼台清交的大學畢業生沒有把他們的研究所填滿? 你真的覺得是研究所考試把他們難倒了嗎?
- 還是有很多的人已經發現考試制度的公平性本身就是失敗的要件, 生命中需要突破的東西根本不在這裡?

對於大家的期許 (cont'd)

- 我也聽到、看到一些實例, 一些業界的老闆可以在一開始就踢掉台清交的畢業生, 現在時機真的很困難, 業界需要的是實力, 沒有實力的話, 學歷光環是撐不了多久的
- 我也聽到一些進了台清交的學長的抱怨: 學習資源獲得的困難, 競爭的慘烈, 所需要付出的適應, 和爭取不到的重視
- 同學, 把精力集中在少數專業科目上, 成功地突破自我學習的門檻, 對你的將來是絕對受用無窮的

對於大家的期許 (cont'd)

- 從小到大, 各種考試只要會 60%, 你就可以順利過關, 記多一點, 看多一點, 練習多一點就會有 80, 90 甚至是 100 分
- 很多時候 100 分又好像是個丟臉的目標, 仔細想想還真的不該是個目標…
- 不過, 學習寫程式如果以會 60% 為目標, 雖然你還是可以順利過關, 但是你自己知道程式很多地方都不會順利運作, 和其它程式合在一起時錯誤一定會出現 (墨菲定律), 你以後會不敢與軟體為伍…其實軟體公司才真的不敢與你為伍…
- 不要把它看成是考試, 但是100% 是唯一的目標

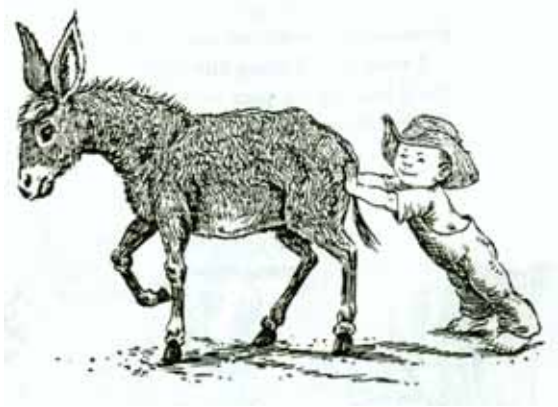
在叢林法則中, 沒有到達
彼岸唯一的下場就是被淘汰

學寫程式的心態

- 當你發現程式執行結果和你預期不一致的時候, 千萬不要假裝沒有看到, 想說下次不會再遇到它了吧! (程式語言的模型很單純, 你一旦遇見它, 就會不斷地犯同樣的錯誤)
- 更不要想辦法隱藏, 隱藏程式裡的 bug 就像說謊一樣, 要用一連串謊言來遮掩, 最後還是會暴露出來的
- 每一個 "錯誤" 或是不如預期的 "程式表現" 都是一個矯正 "程式運作模型" 的機會
- 當你能夠自由地運用程式的語法製作如同你所預期的程式時, 你就算跨過第一個學習門檻了

這個學期裡

- 希望不要



- 也希望不要

- 課程安排比較像

- 後面很多單元裡你會看到有很多的東西需要學習，需要練習… (我不是假設你有無限的時間)
- 學多少是多少，一步一步往前走，所練習的學習方法是最重要的，不要因為吃不到那個蘿蔔而氣餒，真正的牛肉不在這個課程裡…
- 在你以後的課程裡自然會發現了解這些工具以後的好處

運用英文

讀英文書的障礙

- 各位已經在海大上了三個學期的課了, 有沒有真的把某一個科目的英文教科書看過的呢? 該是認真想想的時候了
- 不用吧, 都要自己看的話那何必來上課呢? 上課不就是老師把東西整理好(當然也翻譯好), 讓學生輕輕鬆鬆地學到所有東西嗎?
- 老師以前上課時有看英文書嗎?
- 你究竟是要魚還是釣竿,還是釣魚的方法?

運用英文

- 很多人英文程度不好,但是英文資料的內涵吸收得飛快
  - 美國的教育體系裡 90% 學生是高中畢業, 其中又有大半是不唸書的, 所以他們大部分英文程度的應該都可以說是不好, 至少和你比起來知道的文法比你少, 會拼的單字比你少, 可是需要的時候他們都可以閱讀…很多字猜一下就知道意思了
  - 我看過有學長英文真的不好,可是英文技術資料的吸收程度一流
  - 你們上過的英文老早就可以運用了, 老早就超過許多老美在學校裡學的了, 你看到英文書腦子裡一團混亂可能只是一種莫名的心理障礙

運用英文 (cont'd)

- 英文是一種很精確的語言, 所以英文的技術資料應該是很容易閱讀的, 從英文資料裡重組回來的概念常常是唯一的, 不像中文裡很多字有多種用法, 用在不同環境裡意義常常不一樣, 從文章裡還原回來的概念常常模稜兩可
- 很多人的經驗是: 耐心地閱讀完英文書的前三章, 剩下的章節所需要花的時間就會少很多很多, 跨過一個心理障礙後常常就海闊天空了 (你當然希望在學校裡就可以跨越)
- 你們現在有電子字典, 有 Internet 字典/發音, 有 Internet 上的百科, 已經去除了很多的阻礙了
- 英文句子裡有很明確的語法, 段落也有固定的組織方法

英文基本語法

- 每一個句子的語法都很一致, 就是
  主詞 S + 動詞 V (+ 受詞 O) (+ 副詞 A)
  - 你應該了解到的就是如果是主動的, 就是 S 對 O 作了一些動作 V; 如果是被動的, 就是 O 對 S 作了一些動作, 副詞常常用來表現動作的程度
  - 常常 S, V, O, A 又可以用片語或是子句來取代
  - 所以讀一句話第一件事就是找出 S, V, O, A
  - 例如:
    By using typedef in this way, you can pretend that Structure2 is a built-in type, like int or float, when you define s1 and s2.
  - 困難的地方是常常有很多的子句和片語

英文基本語法 (cont'd)

- 你可以用一招簡化法來分解英文句子
  - 首先你要知道副詞常常是描述動作的程度的,去除它對語法是沒有影響的,影響的是語意的程度; 但是把主詞,動詞或是受詞拿掉就完全不行, 沒頭沒尾的了
  - 例如:
    You can pretend that Structure2 is a built-in type.
  - 上面這一句我只保留了 S, V, O, 重要的是概念還是完整的: "你可以當作 Structure2 是一個內建型態"
- 這樣的分解句子你必須多多練習
- 分解完以後清楚地抓住 "S 對 O 作了一些動作 V" 這樣的概念, 千萬不要拘泥於逐行翻譯…

英文基本語法 (cont'd)

- 更好的地方是科技性的文章中語句常常對應著某些數學或是工程技術的模型或是範例
- 以上面這個例子來說, 整句話的目的是在說明 typedef 的用法, 尤其是在說明下面這個範例

```
typedef struct {
    int x;
    double y;
} Structure2;

Structure2 s1, s2;
```


英文基本語法 (cont'd)

- 你注意看大部分的段落裡像剛才這樣的句子了不起三句, 一頁裡了不起三個段落, 所以你閱讀的文章就變成了

  A K_1 B, B K_2 C, C K_3 A, …

- 注意通常一段文章裡可能只描述一個主體的各種性質, 於是你閱讀的文章就變成

  A K_1 B, **A** K_2 C, **A** K_3 D, …

- 英文文章是很注重邏輯關連性的, 不太會把完全沒有關係的句子兜在一個段落裡, 不常出現

  A K_1 B, **R** S_2 T, **U** V_3 W, …

英文專有名詞

- 在英文書中, 專有名詞第一次出現時幾乎都會以文字清楚地定義, 例如:

  The concept of constant (expressed by the const keyword) was created to allow the programmer to draw a line between what changes and what does not.

- 字典怎樣翻譯 constant 並不重要,重要的是在後面文章裡只要出現 constant, 就代表程式中以 const 關鍵字描述的變數

英文專有名詞 (cont'd)

- 有時也會以一整段文字來定義一個專有名詞, 例如

  You create a new class as a type of an existing class. You literally take the form of the existing class and add code to it, without modifying the existing class. The magical act is called inheritance, and most of the work is done by the compiler. Inheritance is one of the cornerstones of object-oriented programming and has additional implications that will be explored in Chapter 15.

英文段落組織

- 每一段落都有所謂的 Topic Sentence 和 Supporting Sentences
- Topic Sentence 把這一段落最主要傳達的意念點出
- Supporting Sentences 則常常會解釋 Topic Sentence 的立論, 加強論點, 舉例, 細部分析等等
- 閱讀一段英文時, 一定要抓住它的 Topic Sentence 所描述的概念, 確定所有其它的語句都符合這個概念, 如此可以確保沒有誤解
- 隨時問你自己六個相關的問題 **W**ho? **W**hat? **W**here? **W**hen? **W**hy? and **H**ow?

英文閱讀

- 閱讀英文時你可以把還原出來的概念條列在紙上

  ⓪ **A** K_1 B, **B** K_2 C

  ① **A** K_1 R, **A** K_2 S

  ② **A**

  …

- 如此你很快就可以了解並且掌握一個段落, 一個章節所描述的內容

In Summary

- 以上不是要取代你們的英文課…只是要提醒你所會的英文已經可以派上用場了, 不要妄自菲薄
- 閱讀科技英文的能力對你們來說實在太重要了, 有組織地去還原書本裡所描述的知識, 再運用一些想像力, 相信閱讀技術資料應該是很容易的
- 將來如果你能夠擁有撰寫科技英文的能力的話, 更是大大地提昇了你自己的競爭力