中原大學 資訊工程學系

程式語言期末報告

我對語言的看法、想像與期望

資訊三乙 10727211 林彥輝

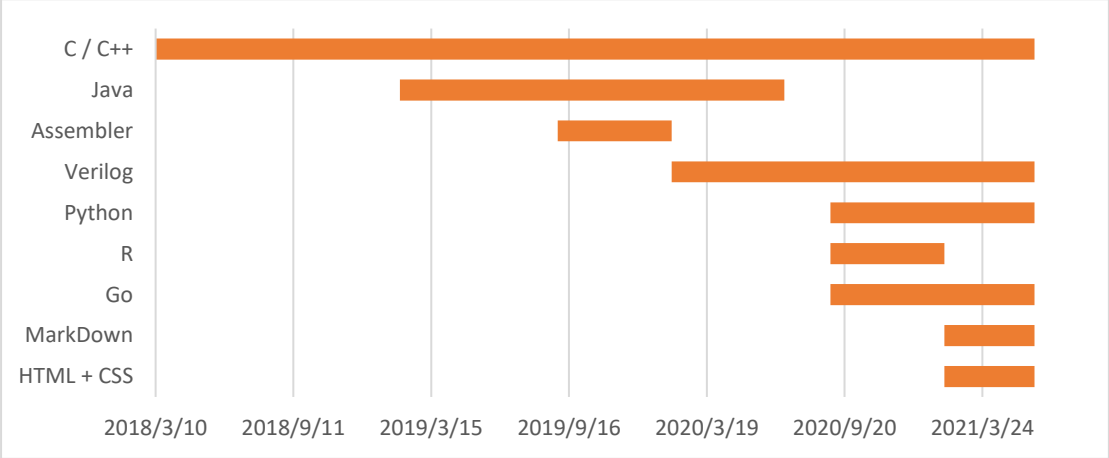
授課老師：夏延德 教授

中華民國一一〇年六月

目錄

一、語言接觸史

在上大學之前，我的程式經驗近乎為零，簡單在高中電腦課碰過電腦課的小認證[1]，以及高中繁星放榜後自己買了一本 C 語言教學手冊[2]，在上課的時候無視正課的存在，使用手寫程式碼的方式在紙上練習簡單的程式，練習內容猶如計算機概論（一）的期中考與期末考題，透過簡單的 Function Call、迴圈，一步步的窺探 Computer Science 的世界。



▲語言接觸史之甘特圖(2018/3/10~)

從 Compiler 到 Interpreter、從後端語言到前端語言、從高階語言到低階語言，在大學的三年中，用了各種面向瞭解機器的運作，也漸漸的有了一套自己看待計算機世界的方式，以下是我在各個階段中思考的幾個問題，以及我的思路歷程。

1.1 為甚麼會有這麼多的程式語言?

語言的目的就是為了溝通，這句話是無庸置疑的，不管是人與人之間溝通的自然語言，或是人與機器溝通的程式語言，又或者是工程師與工程師為了方便所用的偽代碼（Pseudo-code）。

世界上人與人溝通的自然語言約莫五千多種，程式語言的數量也不容小覷，這些語言的基本組成都相似，用程式語言的概念來對比自然語言，套入上課中所述的 orthogonal design principle 來檢驗自然語言，似乎都說得通，最重要的是語言都可以將想法清楚的轉換描述。

1.2 高階、低階？前端、後端？

程式中因為指令集的多寡區分了高階語言、低階語言，高階語言一定就會比較好嗎？在我經由三年寫程式的體驗，答案其實不然，高階語言雖然社群廣大，很容易的將人類的想法表達出來，許多人願意貢獻他們寫的方法造福後人，但相對的，這些程式出現漏洞的機率也高；低階語言指令集簡單，操作上更接近硬體，執行速度因此快上高階語言許多，也因此受到不少人的青睞，但

反觀的，指令集太少的反面缺點就是要寫出一支複雜程式需要耗費相當大的努力。

在大三下學期修到了資訊工業講座課程，因緣際會下接觸到了 HTML、CSS 等前端語言，開始發現老大上課所述的「培養打嘴砲功力」背後的深厚意涵，我們透過寫程式實作例子，卻難以將我們寫的程式透過專有名詞或是有邏輯的敘述將我們的程式架構闡述，意外發現這似乎是中原資工才會有的特色，平常在外與友人閒聊時，聊天內容都是根據話題往下延伸，但在本系中意外發現同學的說話重點會不斷變動，彷彿寫程式般有靈感就中斷跳到另一個重點，這是一個蠻有趣的現象。回到前端語言與後端語言，一般的工程師只需要專注在一個程式背後的邏輯，HTML、CSS 等前端工程師就像個中盤商將後端的程式當成商品加以包裝販賣，實作 HTML 製作網頁之後發現，原來細節並不難，但程式這類的產品還是需要平台包裝才能吸引人，這就是前端致力於的目標。人力網站上缺少的職缺中竟也包含不少前端工程師，意外觀察到臺灣對於各類型工程師的職缺其實是不少的。

1.3 談談我接觸過的幾個語言

1.3.1 我的起手語言：C

簡單的幾個 Data type，加上陣列、迴圈、Function、Struct，以及 Pointer，就能將想法透過程式碼實現，這是我對於 C 最直覺的印象。

這麼好用的語言，一開始只是兩個駭客改寫 B 程式寫給自己用的，所以 Data type 相當簡便，程式的撰寫門檻大副降低，意外造就了 C 語言盛行的基底之一；Compiler 也不會檢查過多除了型別檢查的錯誤，一切都是「夠用就好」。當初的兩位開發人員程式基礎相當優異，因為選擇不做錯誤檢測，所以程式執行速度相當快，這也是 C 的另一個雙面刃。

經過幾個語言下來，我相當喜歡 C 的 Function parameter 中有 call by reference、call by value 的功能，這使得程式的設計更為彈性，若一種程式除了在參數傳遞中除了 primitive type 之外只剩 call by reference 功能，那就必須自行在記憶體中 copy 一份相同的 binding，以達到 call by value 的功能。

另一種 C 的特處，就是 Pointer，「箭頭的起點的那塊記憶體空間所存的內容是箭頭所指向的那塊記憶體空間的 address」，即為 call by reference 的延伸，透過傳址，除了可以更迅速的找到變數的值，另外一個點在於節省記憶體的使用。

問題多的點在於 C 的雙面刃「Error Detect」，C 允許 Pointer 指向的地址做地址的計算，若地址中意外指向記憶體中尚未使用的區塊，極有可能造成程式不正常的運作，但是此類的錯誤是 Compiler 難以發現的錯誤，也造成了指標

成了 Black Hat Hacker 用於鑽漏洞的工具之一。

Garbage collection、「檢測勢力範圍」、強制指標初始化，舉凡此類在任何的開發環境中可能會自動幫使用者做到的，我認為都可以列入改善列表中，將來開投入程式開發的小工程師務必越來越多，各式各樣的人都有可能成為小工程師，是否需要讓使用者自行管制記憶體，或是可以再開發新的函式庫可以自行進行記憶體控管，這是未來新的語言可以好好思考的。

1.3.2 開始啟蒙我的語言：Java

「計算機概論」、「物件導向程式設計」是中原必修課中會接觸到 Java 的課，兩個階段中讓我對於 Java 理解個別不同，前者是 C 到 Java 寫程式思考角度的轉變，後者是教導如何漂亮的使用 OOP 的概念透過 Java 實現，雖然不常使用，但 Object Oriented 的寫程式概念，啟發我開始探討如何將程式寫的更漂亮！

Object Oriented Design 的概念，除了可以將 class 當成 data type 來使用之外，更重要的構思在於 class 內擁有 member function，意味著 class 所 new 出的 instance 有量身訂做的 function 可以使用。搭配 Abstraction 抽象化的概念，描述了 class 與 class 之間階層關係，相當符合我對於人腦思考的運作，是我相當喜歡喜歡的設計手法。

另外老大在課堂上的有提到的設計手法「top-down-design」vs.「OOP design」，在我的角度看是「Error Handling 的差別」，top-down-design 設計出來的 Function，其 Error Handling 的設計要相當全面，因為難以阻止會有不經意的各種型別，尤其是在設計份量越大的程式時，反觀 OOP design，其 member function 理想中是為了 object 量身打造，因此 Error Handling 的部份可以不必如此謹慎。

最後，Garbage collection 機制是我相當欣賞的概念，減輕了開發者在設計程式的負擔，犧牲一點執行的效能，換取 Writability、Reliability 我認為是相當值得的交易。

1.3.3 自學的語言：Python

套件豐富、AI 是 Python 崛起迅速的原因，Python 的語法比較不嚴謹，對於新手相當容易上手，現在的多數 Project 中要我選擇語言開發，Python 會是我的第一選擇，Python 的範例程式相當豐富，有社群特別製作了一系列的範本給開發者使用[3]，對於有一定程式基礎的人來說，自學 Python 的所需的陣痛期不會太長，Python 可以說是「高階語言中的高階」，有許多的機制方便且讓使用者使用起來相當直觀，如 list 功能，若使用恰當，會比 vector 好用許多，因為在 list 中的內容物可以不必限定統一 data type，但這個特性也可能

是 Bug isolation 的缺點，前提要建立在開發者要有充足的程式基礎。

Bug isolation 是 Python 中最玄學之處，Python 的語法難以捉摸，型別會自動轉型的特性，Dynamic Binding 的特性，讓 IDE 顯得更加重要，一個良好的 IDE，透過 Debugging Mode，會更容易找到程式的 BUG。

1.3.4 我認為相當特別的語言：R、Verilog

不管哪一種程式語言，要成為主流就要有與眾不同的特點，才會吸引社群投入貢獻，例如機率與統計課接觸的 R 語言，雖然上課時只是簡單照著範例程式循序執行，但光是如此就能體會到 R 語言之分析、統計、繪圖之強力之處；Verilog 也是相當特別的語言，其提供的特點如其中文名稱「硬體描述語言」，透過 reg 型別與 wire 型別實際模擬線路訊號在邏輯閘傳遞的模型，也從中體認到「Finite-state machine」的概念，Verilog 強大的訊號模擬功能也漸漸成為硬體授課的最佳教材。

二、教育的領域適合使用甚麼語言做媒介

在我上錄取大學之後，第一件事情就是搜尋「中原資工」，就看到了採訪[4]的報導，以及影音平台上的影片[5]，在還沒入學之前，就知道老大是個有自己一套教學理念且德高望重的老師。

2.1 老大的教學方式

「只要你肯學，我肯定把你教會」，這句話是老大說過且一直銘記在我心的一句話，從 Visual-C、CAL、PAL，會發現這一系列的作業，都是訓練，至於怎麼訓練？就是「實作」，程式相當重視實作，如果只是一味的在 Text Editor 上寫 Program 而不去執行成為 Process 檢視結果，那這些 Program 就只是空談。這也是激勵學生繼續寫程式，跨過眼前的障礙的方式。

「我盡量教少，但我要求你們全部都懂；我教的指令不多，但我教的這些就足以搞定所有的程式！」，在老大上課曾經提到[6]，只要有 if-then-else、iterative、sequence programming，就具備了一個程式最基本的功能，意味著剩下的指令都是「方便」使用者而開發出來的指令，可以讓使用者更專注於開發其餘的功能，而不必冒風險刻一個指令使用。而對於初學者而言，一次接觸過多的指令反倒是種負擔，且程式寫起來綁手綁腳，倒不如使用一個方法，且用至專精，有餘力再去研究其他的指令。以中原資工中最好的例子，就是「for 迴圈」，事實上，老大只有教如何寫 while 迴圈，Visual-C 中也只有教 while 迴圈，但在老大談到 for 時，只淡淡的說了一句，「你們早就會了！」，這個道理到寫 PL Project 時才深深的領悟。

老大的寫程式哲學與教導寫程式的哲學，成效可以透過大學排名中體現，

可以說排名輕鬆衝擊四大，這個傳奇教法相信在老大有生之年還能繼續締造傳奇。

寫程式，只是一種練習，透過不斷的練習，漸漸的建立自己寫程式的一套哲學，接著用更漂亮的方法解出同樣的問題，讓程式更加堅固減少漏洞，每個程式設計師，就像是一個制定法律的大法官，社會並非一成不變，在社會有創新趨勢的同時，要有跟進的法律來確保這些趨勢可以照著正當的方向發展，同時，抵禦背後鑽漏洞搞破壞的臭蟲。

2.2 什麼樣的人適合用甚麼樣的語言？

「我想學程式，我該從什麼語言學起？」這想必是一個程式初學者最經典的問題，以我的案例，我是先透過參考[7]，網站下方的「擅長工具」，找到C/C++、Java，所以才有了第一步的方向，那其他人呢？我的推薦會有兩個方向：研究為導向（目前尚未遇到）、目的為導向（他校之同學有作業危機，用優渥獎品請我開示並做法）。

以研究為導向，我會傾向先學 C，之後再轉 Java，一部分是 C 是我的基礎，另一部分是 C 的概念相當基礎，且只要照著範例程式一步步改寫做題目，就能擁有相當不錯的基礎，等至擁立 Iterative、Function、Recursion、Struct、Class、Pointer 概念之後再轉至 Java 學習 OOP 的概念，屆時任何的想法要透過程式付諸實行想必都不成難題。

以目的為導向的人，想必時間是他們最重要的籌碼，因此 Python 是我推薦的利器，Python 可以很簡單的描述人類的想法、函式庫豐富、範本多[3]，不須嚴謹的語法，一個 print()就能輔助完全沒有程式基礎的人輕鬆了解指令、程式碼像是一個自然語言的語法，大大增加了程式的可讀性，上手相當容易，不少學習 AI 的人第一個語言就選擇 Python，無疑成為社群成長最迅速的語言。

三、寫程式的哲學

3.1 如何學多個語言、如何寫出漂亮的程式

學寫程式是個練功的過程，想一次做好全部的事情近乎不可能，例如學習不同語言為例，不太可能將所有語言一氣呵成學到淋漓盡致，儘管有許多的概念是相通的，如 primitive data type、OOP、iterative、function，但不同的語言就有其特別之處，一次學太多的語言就可能會像國高中階段考試般，無法真正將學到的東西學以致用。依我的觀點，最好的方法就是先將一個語言練到相當精通，接著再去換個思考方式去學其他的語言，「用心感受兩者語言帶給使用者的差異」，進一步擁立自己喜歡的 coding style。

如何漂亮的寫出程式，我的答案是「多看他人的程式碼」，事實上「漂亮」是一種非客觀的形容詞，因此可能指的是程式行數少、執行速度快、組織架構

相當完整，這類的經驗值累積完全倚靠觀察，每一種漂亮的定義要做到最好都是一種學問。

3.2 一個人到底要知道多少才有資格寫程式

這個問題相當哲學，寫程式是一個不斷與計算機打交道的過程，事實上，任何寫程式碰到的問題都是在幫助你更了解這個語言的運作及熟悉電腦的運作，當今世界網路發達，任何的問題、任何的領域都有人在奮力研究著，只要有心，任何問題都不再是問題，至於一個人到底要知道多少才有資格寫程式？我會如此地回答：「追求這個問題的答案，就是這個問題的答案」[8]。

四、新的語言

在研究專題時，接觸到了 Go 語言，有人形容 Go language 是「C + Python」的合體，剛好就在某個晚間夢到了這麼一個新的架構，即「C + Verilog + Python」。

Verilog 的範例程式碼(4bit 加法器)：

```
module AddSub4(a, b, sel, s, cout);
  input [3:0] a, b;
  input sel;
  output cout;
  output [3:0] s;
  wire [3:0] b_temp;
  wire [2:0] cout_temp;
  //
  // the code body is here...
  //
endmodule
```

在 Verilog Code 語法的宣告中，參數的型別是在 2~6 行決定的(input、output 是邏輯閘的 IO 訊號，default 型別為 wire)，這是我覺得可以拿來改造的想法材料。

Python 的範例程式碼(Dynamic Binding)：

```
def sample_function(type, binding):
    if type is 'string':
        print( str(binding), 'is', type)
    elif type is 'int':
        print('( 100 /', str(binding), ') is :', str(100//binding))
                        # divide by int operater is : '//'
```

```
if __name__ == '__main__':
    sample_function('string', 'WhySingleQuote?')
        # print : WhySingleQuote? is string
    sample_function('string', "WhyDoubleQuote?")
        # print : WhyDoubleQuote? is string
    sample_function('int', 20)
        # print : ( 100 / 20 ) is : 5
```

C/C++的範例程式碼(function overloading)

```
string add( string a, string b ) {
    return a + b ;
} // add()

int add( int a, int b ) {
    return a + b ;
} // add()


int main() {
  cout << add( "AAAAA", "BBBBB" ) << endl ;   // AAAAABBBBB
  cout << add( 10, 20 ) << endl ;             // 30
} // main()
```

新的語言

Python 中的 Dynamic Binding，使得每一個變數的型態要等到 run-time 時才能決定，新的語言中將以 C 為基底，將 Dynamic Binding 的概念略做修改成 Static Binding，透過 Verilog 中宣告變數型態的想法，新的語言將支援 Function 參數允許多型別的引入，但必須在 Function 中定義窮舉支援的型別(即取代 function overloading 的想法)，如此一來，支援型別檢查，減少 Bug isolation 的機會。

另外，將基於 C 語言並新增 C 所不支援的功能，如 Garbage collection、repeat-until、陣列的索引檢查。

語言範例

```
function add( a, b ) {
  declaration {
    int, string a ;  // all variable type shall be Declaration here
    int, string b ;  // for compiler to compile the variable's data type
  } // declaration()

  // If have two or more data type, Exhaustive all possibilities
  if ( typeof(a) == int && typeof(b) == int )
    return a + b ;
  else if ( typeof(a) == string && typeof(b) == string )
    return a + b ;
  else
    throw exception ;
} // add()


int main() {

  cout << add( "AAAAA", "BBBBB" ) << endl ;   // AAAAABBBBB
  cout << add( 10, 20 ) << endl ;          // 30
// ------------------------------------------------------------------
  repeat {
    work++ ;
  } until(I'm down) ;     // opposite to (do-while)
// ------------------------------------------------------------------
  return 0 ;

} // main()
```

五、我眼中資工的未來?

人類為了透過程式碼可以模擬腦中的想法，舉凡在高階語言中使用 primitive data type、if-then-else、iterative 等運算工具，以循序邏輯設計出程式來要求機器做到人類的想法，隨著新課綱[9]的釋出，「寫程式」反過來成了訓練邏輯的手段，排除那些對於程式課程（Coding 或是程式邏輯遊戲[1]）沒有興趣的人，真正寫程式帶來的影響容易在生活中體現，如演算法中探討的各種經

典問題、作業系統中的 CPU Schedule，與其說程式讓新生代的小工程師生活變得更循序，倒不如說透過生活中的各種案例，思考這些案例透過程式碼該如何解決或做得更好，說不定會有新的創意可以突破現代技術的框架。

「現代的 AI」(影像辨識）的強度受限於人類對大腦的研究，在醫學專家尚未真正理解大腦的細部運作之前，AI 的大腦（Model）就只能由相對大腦簡易的演算法來模擬大腦的分層組織，那人類與「現代 AI」的差別在哪裡？舉一個阿珠與阿花的故事[10]，假設有兩個女朋友：阿珠、阿花，阿珠長得漂亮、知書達禮，但家境背景複雜，娶了有可能招惹是非；但阿花爸爸是富商、黨政關係良好，但阿花長的不盡人意，若是 AI 來選擇兩者當老婆，那答案可能會有兩種，但人類就不一樣了，可能會有"有沒有其他選擇"的答案，這就是人類無法模擬的特性「人類會思考」，換句話說「創新」也是 AI 無法使用演算法得到的答案。

世界因應疫情，正推出不少的數位轉型，將許多實體的服務漸漸佈署到網路上，以金融業為例：推動無現金交易帶動許多網路銀行的服務，讓銀行的服務可以帶著走。在教育界為例，加速雲端視訊會議的技術，各家軟體公司研究如何處理大併發的問題，讓使用者體驗不受限。以上的問題顯示著軟體的世代即將來到，任何的服務都可以透過軟體服務完成。

有了這麼多的技術需求，代表底層的工程師的數量會越來越高，甚至程式會成了每人必備的工具，經過了二十年，英文成了第二外語，未來的三十年，相信程式將成未下一個第三外語。

參考資料

[1] *HourOfCode*. https://hourofcode.com/hk/learn

[2] 洪維恩(民 96)。C 語言教學手冊（四版)。台灣：旗標。

[3] *工程師好用資源來了！超完整 Python 查詢表，程式碼複製貼上不用自己寫*. (2019, July 16). https://buzzorange.com/techorange/2019/07/16/github-python-resources/

[4] 中原資工班刊◎你所不知道的夏老大～夏延德教授專訪. (2007, April 2). https://jackyreading.pixnet.net/blog/post/3651299

[5] 陳柏豪. (2017, September 21). *106 中原資工一乙計概課上課實況紀錄(請打開字幕)*. YouTube. https://www.youtube.com/watch?v=lWRw5pH6M6s

[6] Buneman, Peter & Davidson, Susan & Suciu, Dan. (2000). Programming Constructs for Unstructured Data.

[7] *私立中原大學資訊工程學系（中原資工）—104 升學就業地圖*. 104 人力網站.
https://www.104.com.tw/jb/career/department/view?degree=3&sid=5060000000&mid=520114

[8] 追求這個問題的答案，就是這個問題的答案：Tedxtaipei. (2013, October 31). *生死的智慧：柯文哲 (Wen-Je Ko) at TEDxTaipei 2013*. YouTube.
https://www.youtube.com/watch?v=N0zhdMwD2Z8&t=808s

[9] 課程及教學研究中心. 新課綱「程式設計」，學邏輯解問題.
https://epaper.naer.edu.tw/edm.php?grp_no=2&edm_no=134&content_no=2672

[10]臺大演講網. (2015, September 26). *Understanding Human Behaviors from the Perspective of Science | 柯文哲 台北市長 | 臺大演講網*. YouTube.
https://www.youtube.com/watch?v=UDAc5M4nTk8