我對(與寫程式相關的)語言的看法、想像與期望

資訊三乙 10727219 許志仲

1. 曾經接觸過的語言

Visual Basic

在我高中的資訊課裡老師的教材中有教我們學習使用Visual Basic語言，這應該算是我第一個學會的程式語言，VB是由微軟基於Basic所開發的程式語言，是第一個支援視覺化介面設計的程式語言，它的特色有語法簡單易懂、交談式解譯與執行、自帶GUI等，在當時我們需要安裝Microsoft Visual Basic來編譯VB程式，最特別的就是我們需要先手動用拖曳的方式來創建自訂的視窗，像是新增Buttom、Label等物件，系統會在後台自動幫你生成對影的程式碼，之後再進行coding，縱使後來我還是發現Microsoft Visual Basic提供包含主控台視窗(或者說我們現在常用的黑底白次命令頁)、Windows Form…等模式，但我當時還是以為所有程式語言都是用這樣先自訂Windows Form的模式在運作的。

在我印象當中，VB是個相當簡單直接的程式語言，它的語法大部分都相當簡單直覺，讓我們這種高中生初學者很容易了解程式碼的意思，Microsoft Visual Basic在編譯過程中也會指出語法錯誤的程式碼，並提供相對應的錯誤說明、提供修改方法，撰寫過程中還提供自動選字功能，對我們來說算是相當方便的。

說道VB的缺點，正因為它的簡單，導致很容易造成漏洞，功能也不向其他語言來的強大，導致越來越多VB使用者轉向改用C#，微軟.NET開發團隊也在去年宣布不再將VB當作獨立語言發展，但我依然認為VB是很適合初學者學習程式語言的工具。

C/C++

C/C++應該是世界上最廣泛被使用的語言(之一)，具有高效、靈活、功能豐富…等特性，也是我大學踏入中原資工之後學到的第一個程式語言，C語言最初由貝爾實驗室基於B語言為基礎開發，更後來的C++為在C語言的基礎上，新開發得”包含class的C語言”，有點像是C語言的增強版，後續又持續增加operator overloading、exception…等功能。

說到一開始上計概時，因為CAL系統無法複製貼上的緣故，我們偶爾必須另外自己測試自己的程式撰寫邏輯，當時系上的同學都會使用Dev C++來寫程式，不過Dev c++有缺乏維護版本老舊的問題(聽說已經有新的開發團隊接手維護更新了)，現在大家比較常使用VS Code來寫程式，說到這個就要提到C++ 11 這個C++語言的更新版本，在C++11中提供了相當多的功能，像是用來確保記憶體管理的智慧型指標、lambda函式(不確定是否與scheme中的lambda function功能目的一樣)、如stoi()之類的型別轉換function(如果PAL系統能夠支援的話Project應該能夠寫得更好的)。

縱使擁有廣泛的擴展性和豐富的功能，說到C/C++語言我覺得缺點還是在於可以直接到記憶體位置進行修改的部分，這樣導致使用者很容易駭進系統，導致老大會在PAL系統裡加入多項檢查限制？，進而導致我在寫PL project時，送出程式碼後需要等待很久的時間，還要登入bbs申請增加執行時間，一定程度降低了我檢查程式的效率。

當然我認為PL系統在風格檢查處理的機制裡也有相當不錯的地方，在PL project裡，我嘗試使用了自訂義struct Token的指標的陣列，在送進系統時被風格檢查擋了下來，原來在系統中是不接受Token \* tokenarray[] ; 這樣的操作的，後來我使用的解決方式是先對Token的指標進行typedef，額外命名成TokenPtr，才能正確的以TokenPtr tokenarray[] ; 的方式正確執行而不被系統擋住，這樣子程式也比較容易閱讀而不混亂，不然以C/C++這種對記憶體位置存取相對寬容的環境來說，非常有可能因此搞混而發生錯誤。

Python

不知道從甚麼時候開始，機器學習成為資工系專題題目的首選，系上幾乎每個教授都在做機器學習相關題目，而Python語言，憑藉著它嚴格一致的特性，以及numpy, scikit-learn…等對於科學計算相當方便的工具，成為了機器學習使用程式語言的首選。

第一次使用到Python寫程式應該是在大二工程數學的時候，我們必須要Python實現一些微積分函數圖形的繪畫，得益於numpy強大的計算能力，以及matplotlib的繪圖功能，我們能用python輕鬆地畫出很多複雜的圖形，雖然在這之前我們只學習過C以及java的語法，但是根據系上教授的說法，這是我們應該自己學會的。

在以往學過的語言，如C、Java中，變數再宣告過程中都是需要先給予固定的型別的，而Python是Dynamic Type Binding，可以在程式執行過程中任意變換type，這是我認為python最特別的地方，一開始不習慣時，常常需要一直注意這個變數到底是甚麼型別，在執行程式時總是一直跳出error訊息，說這個type沒有這個功能，我覺得這個功能有利有弊，優點是可以根據需要任意改變變數型別，不用再額外宣告同樣目的但是不同type的變數，像是在數字計算的程式當中，可能要同時宣告intnum、floatnum之類的變數(如同PL OurScheme程式一樣)，缺點是如果不是寫程式的人本身，要看懂程式就會變得相當複雜，因為不確定程式執行到現在這個變數到底是甚麼型別，就算是撰寫者本人，也有可能在程式行數過大時自己也搞混。

另一點我對於Python語言相對困擾的點是對於他多樣的函式庫，雖然說Python得益於這些函式庫擁有非常多強大多元的功能，但是在這樣的狀況下，常常導致不同工具之間版本的相容性的問題，我們在專題實驗時就常常遇到類似的問題，網路上參考過來的實作方法常常需要搭配很多套件，不同的套件還有各自限定的版本，太舊或是太新都不行，因此常常需要切換虛擬環境去安裝相對應版本的套件，甚至因為Python有太多套件的關係，有時候不同的套件還有可能互相衝突不能一起安裝，對於我們在做專題實驗題目來說造成了不小的困擾。

Rust

再來談到一個我沒有使用過的語言Rust，最初是從就讀那所雖然程式能力跟我們差不多但是比我們會打嘴砲的中央大學的高中同學聽來的。

相較於其他語言，Rust屬於較年輕的語言，從2010年作為Mozilla官方專案出現，到現在也只不過11年的時間，作為一個新興的程式語言，Rust具有不少高階語言的特性，但仍具有接近 C++ 的效能，其中我認為最大的優勢在於它的安全性，在C/C++裡，我們可以任意的讓指標指向任何記憶體，這種機制雖然自由，但也伴隨著安全性的風險，但在Rust的編譯器中，它會將危險的記憶體操作擋下並當作程式的Error，不過其餘無害的操作仍然可以執行，這樣子雖然導致程是在編譯時會導致出現比較多Error，但好處是程式實際在執行時比較不容易發生錯誤，就如同PAL系統一樣，我們常常在外部執行自己寫的程式時都沒有遇到Error，但是將程式丟進PAL系統時卻常常出現很多complie error(雖然說之後的風格檢查更麻煩就是了)。以目前來說，因為Rust屬於比較新興的語言，它的擴充性與功能相較於其他成熟語言還有不小的落差，而且Rust屬於語法相對複雜的語言，對於初學者可能相對較困難，但我認為它依然是個有趣的語言，它宛如跟PAL系統一樣的安全檢查讓我頗有共鳴。

Whitespace

最後講到一個我在苦惱該怎麼寫完這份報告時在網路上查到的特別的程式語言Whitespace，在大部分的語言當中，whitespace都是被我們以及電腦所忽略的，除了像是Python會以tab字元來表示在function，判斷式裡，又或是在PAL系統中，風格檢查會強制要求我們使用2個空格進行縮排，除此之外在大部分的情況下，whitespace字元對我們來說，除了可讀性之外都不是太重要。

但是在Whitespace語言當中，whitespace可說是最重要且不可取代的要素，在Whitespace程式碼中，所有的指令都由whitespace字元組成，包含space、tab以及換行字元，實際上的空白、tab、換行字元則分別以S、T、L代替。另一個特別的點是Whitespace語言的編譯器在編譯城市時會自動忽略文字資訊，所以如果要在程式之中加註解的話只要直接輸入就可以了。

要說到這個Whitespace語言最適合使用的時機，應該就是在加密工作的場合了，在沒有特別意識到的情況下，Whitespace程式碼看起來完全就是一坨空白，又或者可以在其中加入文章，畢竟編譯器在編譯時會自動將這些文章視為註解的一部分，對於那些需要保證機密及安全性的程式工作來說，我認為Whitespace應該就是最佳的選擇。

1. 有興趣的領域

如果要說選擇就讀資工系的理由，除了打電腦遊戲之外，應該就是網路安全領域了 吧！因為我覺得很帥，不論是越過系統防護破解資料、或是抓出系統漏洞防守敵人的攻擊，用想的就覺得是一門很厲害的學問。

在網路安全領域當中，程式能力應該也是相當重要的，我覺得在檢查系統程式漏洞、發現惡意code的處理方面，跟我們平常在寫程式時debug的作法是差不多的。

一開始我以為在網路安全方面，大家常使用的語言應該會以網頁、數據庫相關的程式語言為主流，像是JavaScript、SQL…等語言，但是根據google到的資料來看，Python有在網路安全領域也不乏有許多使用者，這個發現蠻出乎我意料的。

Python 在人工智慧盛行的最近幾年間儼然成為了最熱門的程式語言之一，很大一部分的原因就是因為它的功能廣泛且簡單易學。在網路安全的領域當中，常常在短時間有變化萬千的問題，很多既有的技術及功能可能在短時間內無法因應快速的變化而變得無用，因此大家才會轉而使用簡單且多功能、擴充性強大的語言，期望直接以自我的程式設計能力，因應不同狀況問題編寫對應的功能，而Python正好就是這些要求的首選語言。相較於其他的語言，我認為Python的優勢就在於它龐大的擴充函式庫可以幫助網路安全工程師在解決問題時找到適合的功能，而且網路安全常常牽涉到密碼學相關，不論是解碼、編碼的應用，我認為適合進行大量數據科學計算的Python相當適合，Python語言本身簡單開發容易的優勢也很容易讓資安工程師迅速為新出現陌生的漏洞及問題進行反制，又或者是讓擁有專業技術但程式能力不夠強的工程師較快速的學習並實踐。

1. 理想的語言

我覺得一個程式語言的首要條件應該要是平易近人，畢竟人類一直以來都在嘗試用電腦做原本屬於人類在做的事，程式碼應該也要人性化一點才是，就以我們目前常用的C語言來說，我們常常會忘記在if或while的判斷式裡，判斷a是否等於b的值時應該要使用==而不是=，如果使用了=，除了無法正確的判斷條件之外，還會改掉變數原本的值，進而導致程式錯誤。但我認為，以現在的技術來說應該很容易判斷程式碼是否處於判斷條件才對，因此應該可以讓=符號同時擁有assign與比較的功能。程式碼的整齊度也是影響可讀性相當重要的一環，我覺得像是Python中用tab縮排表示的方式就很直觀，一眼就能看出在程式碼之中的上下層關係，在我自己的語言裡，這應該能代替原本用大括弧匡列內層程式碼的表示方式。

另一個我想加入的，是在程式語言中融合靜態連結和動態連結兩者，在某些程式之中，我們可能某些需求需要改變變數的型別，縱使靜態連結有它的好處，能夠幫助電腦在執行程式時妥善的依照編譯時變數的型別分配記憶體空間，但我認為這樣子在某些狀況下不太方便，以現在的技術來說也不是以前那種記憶體寸土寸金的日子了，電腦應該要可以更自由的分配記憶體空間給變數，讓記憶體配置更為靈活而不死板，在程式執行時應該要可以因應程式撰寫者的需求進行變數型別的轉換，以PL的project來說，在做加減乘除相關指令時，我們常常需要在整數跟浮點數之間轉換，當然我們也可以使用C++ class中繼承或是多型之類的方法，讓我們的Token只帶有對應的整數或浮點數型別變數，但是在實際做計算時還是有些不方便，我自己目前的做法是在計算時全部轉為浮點數去做計算，直到回傳值時確定它至始至終是個整數再額外把結果轉成整數回傳。比較大的問題在於，在一次次的轉換之間，難免出現浮點數的誤差，可能到導致計算出的結果不一樣，尤其是在後面的題目裡，有些過大的數字計算會超過四捨五入的範圍，造成整數結果直接差1，導致答案不符預期，又在C/C++裡，每個人對於字串型別轉數字的做法不同，甚至是編譯氣的版本也不一樣，如此一來就會導致每個人的輸出結果不同。如果說在程式語言中可以提供額定標準的型別轉換，讓我們可以任意變更變數的型別，又可以確保該變數的值是固定不變的，那肯定是相當方便，只要簡單一行指令，就能字串轉integer，integer轉float，那肯定是我理想的語言一定要有的功能。

1. 那些我寫PL的事

可能因為我不是很會打嘴砲的關係，看起來應該是寫不到規定字數了，整好在寫報告當下我已經寫完PAL project了，因此我想要分享一下我在寫project的感想。

實際寫完之後我認為四支project的難度其實真的不難，只是要處理的指令和錯誤訊息相較於以往寫的程式多了不少，考量到這學期我還有系學會的關係，四支project總共也就花了兩個多月的時間寫完。我認為把整個程式分成四個部分來完成算是蠻特別但是很有用的做法，可以引導我們在寫程式時有效的劃分不同階段程式需要完成的作業，我們也能順著題目有效的將程式碼切成各個實做不同功能的區塊，要寫完這部分的要求並且答對才能進行下個階段，對我在debug時起到相當大的作用，大部分時候遇到錯誤時，我幾乎都能先肯定其他地方已經ok了，只有新增加的程式碼還有問題需要處理。

在寫Project時遇到比較大的困難，應該就是在了解這個指令到底應該做甚麼？發生這個Error應該怎麼處理？這個問題在Project1, 2, 4的時候並不太困難，因為說明文件在這三個Project中對於各項指令和錯誤訊息的說明都非常詳細，就算是我在Project1一開始卡隱藏測資時，依舊能從說明文件中的範例和文字敘述中交叉找出我自己程式的錯誤並加以修改，在寫程式的過程中算是相當順利的。但是在Project3中就遇到了相當大的困難，Custom Function和Lambda倒是沒有遇到太多問題，額外處理區域變數即可，但是在Project3中新增了很多No Return相關的Error，這些錯誤的處理方式我自己覺得在說明文件中並沒有太詳細的說明到，也沒有足夠的範例讓我們來驗證答案，因此在Project3後面的題目中我必須從測資中去推敲可能的答案，這樣做的結果就是卡了隱藏，後來花了很多時間嘗試將所有可能性再檢查並修正後才通過，我也才能了解到這些錯誤訊息的原理，我認為這是再OurScheme全部四支Project中最困難的一段。

總結來說，我認為寫完這支Project對我來說幫助還是很大的，不但可以練習如何將非常大支的程式解構成小問題，並且各個擊破(這應當是程式設計的核心)，也能在解析scheme指令時，了解到電腦對於程式碼的判斷及處裡方式，讓我有種程式功力更加提升的感覺，對於能夠在夏老大退休前順利修完PL(目前看起來應該可以)，個人感到非常慶幸。