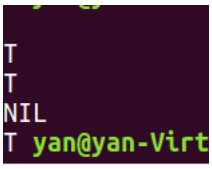
程式語言 HW1 作業說明

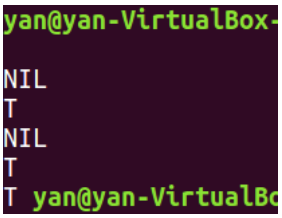
資訊三乙 陳華嚴 F04056154

1. 執行環境：Ubuntu 16.04 shell
2. 執行步驟：sbcl --script <檔名>
3. 完成題目：**problem1 全 + problem3**
4. 執行結果：

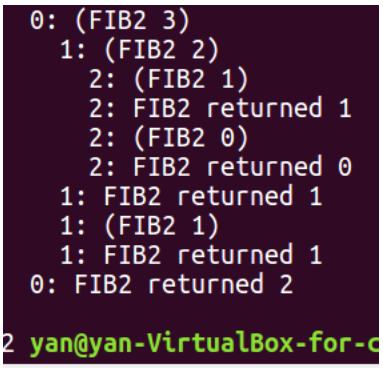
【Problem 1.1 prime】



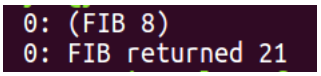
【Problem 1.2 palindrome】



【Problem 1.3.1 Fibonacci1】



【Problem 1.3.2 Fibonacci2】



【Problem 3 Diff command】

5. 程式碼說明：

- Problem 1.1 **sbcl --script prime_number.lsp**：
  如果 x==1 的話，回傳 nil；如果 x==2，回傳 t；剩下的情形就是其他的了，我設的迴圈範圍是從 2~（x/2）取下高斯，其實後來想到用 2 取到 sqrt(x)就可以了（離散數學有教），而使迴圈跳出來的方式就是寫 never，即若 input 的數字一可以被迴圈所迭代的數值除以後所得到的餘數為 0，則會回傳 nil，而過程中若都沒有得到的餘數為 0 的話，即表示就是質數，就會傳 t。
- Problem 1.2 **sbcl --script palindrome.lsp**：
  一開始我設了一個全域變數 reverse_list，這是用來等等放顛倒的 input，我的作法是直接把一個 list 倒過來，再比對原來跟倒過來的，如果還是一樣就代表符合回文，會回傳 t；不一樣即回傳 nil。
- Problem 1.3.1 **sbcl --script fib1.lsp**：
  直接用一個 if 來判斷，如果小於 2，就回傳自己（因為等於 1 時回傳 1，等於 2 時回傳 2）；如果大於 2，則就會執行 n-1、n-2 為參數不斷遞迴下去直到基本條件達成。
- Problem 1.3.2 **sbcl --script fib2.lsp**：
  這道題目我是參考課堂投影片的，tail recursion 和一般 recursion 的差別即在 stack，tail recursion 因為都直接在參數裡算好值，所以可以一直傳下去而不用像 direct recursion 一樣要瘋狂疊 stack，也因此用 trace 的時候可以發現很精簡的 trace 過程。
- Problem 3 **sbcl --script diff_command.lsp**：
  這超難的，從開始到寫完花了 10 幾天！一開始我是上網查到好像可以用 lcs 演算法做，可是因為不熟悉 LISP 所以失敗了(程式碼下面有註解掉的那幾十行就是失敗之作)後來我先判斷檔案一和檔案二是否有相同的行，如果相同的行為 0，則就分別抽出檔案一和檔案二的行數交疊-+-+輸出；若是相同的行數大於 0，則用檔案一的每一行去比較檔案二的每一行，如果有一樣的話，那就會輸出檔案一在檔案二那行之前的”所有”行數，並且都在前面加上”+”；倘若檔案一的該行比較檔案二毫無相同之處，則就會輸出檔案一該行並且在前面加上”-”表示是檔案二沒有的程式碼，一樣的話就直接 format 輸出。而我的檔案一和檔案二的內容使用的是助教作業說明中的那兩個檔案。
  我使用了大量的全域變數(因為用 let 設區域變數我會頭昏眼花)，然後用開檔輸入的方式使檔案可以存成一個 list，分別放進 list1 和 list2，在這兩個 list 裡面的 atom 就是程式碼的一行。然後在做的時候發現 LISP 有很多實用簡便的函式，例如 list-length 可以算出 list 有幾個 atom，然後 nth 可以抽出 list 中指定的 index 的 atom，於是就利用 nth 對照。我設定了一個雙重迴圈(其實本來要用 do 或者兩個 for 寫但是一直一直參不透 syntax

所以放棄）中間的演算過程就是上面所述，值得一提的是，可以利用設定 block 名字的方式，使很裡面的迴圈得以直接跳到那個 block。寫到最後的時候發現程式好像一直不會自動結束，所以我使用了 end_mark 標記是否已經讀到最後一行了，是的話就使用 cl-user::quit 直接結束。