C Language Learning Platform for Primary Students

Chi-Chou Kao[1,*], Wei-Zhen Guo [2,*]

[1,2]Department of Computer Science & Information Engineering, National University of Tainan, Tainan, 70005, Taiwan

Abstract

In this paper, we propose a C-language learning platform for primary students includes a C-language learning platform, an interpreter and a translation terminal, and a C language learning platform. When the user confirms the identity through the login, the user can enter the C language learning platform. After learning the program learning unit or the question unit, it will be sent to the translation batch to perform translation, compilation and correction. After the user understands the error, the system can be used repeatedly until the learning is completed. The invention translates error messages into easy-to-understand chinese, and there are a large number of non-choice questions for users to learn, so it is suitable for primary students to learn C language.

Keywords: debug oriented, program design, platform, test library, compiler.

---

* Corresponding author: cckao@mail.nutn.edu.tw

DOI：10.3966/222344892019100090205

中小學生 C 語言學習平台

高啟洲, 郭維桓

國立臺南大學資訊工程系

摘要

本論文提出一種中小學生 C 語言學習平台，包含有一身份辨認登入端、一翻譯批改端與連接於兩者之間的一 C 語言學習平台，當使用者經身份辨認登入端確認後，可進入 C 語言學習平台中的程式學習單元或題庫單元學習後，送至翻譯批改端執行翻譯、編譯和批改等動作，使用者了解錯誤所在後，可反覆使用本系統學習直到學習完成為止；本平台因會將錯誤訊息翻譯成淺顯易懂的中文，並有大量的是非選擇題庫供使用者學習，因此適合中小學生學習 C 語言之用。

關鍵詞：除錯導向、程式設計、平台、題庫、編譯器。

1. 緒論

因為軟體的快速成長，使得每個學生在學校幾乎都必須上程式設計的課程，但許多學生經常面臨一個問題，常因粗心所造成的語法錯誤或錯字，而出現錯誤，卻又因為看不懂錯誤的訊息，因此感覺寫程式是太困難了，最重要的是，從編譯器上得到的錯誤訊息都只告示語法錯誤和錯誤發生的地方，使得學生不能有效地利用來偵錯，因為一般來說藉由這些錯誤訊息來偵錯是需要經驗的，但是不幸的，這些訊息都是英文的，所以導致英文能力較差的學生學起來更困難，程式語言就像是 20 世紀時的英文一樣，在未來世界中是理解與創新必要語言，以後不管是工業 4.0，又或者是物聯網，都是需要程式設計師來寫這些應用。因為硬體方面已經遇到瓶頸，但是軟體方面能發展的還有許多，根據美國勞工統計局的計算，從 2010 到 2020 在美國電腦程式設計師的職缺數，預計大幅度成長 30%，同時，其他工作的成長預估有 14% 而已。所以如何培養出一個好的程式設計師，是非常重要的，所以我們就必須從基礎層開始做起，初學者的起步是最重要的，有可能剛開始寫程式時就馬上遇到困難，寫完程式想要去修改自己錯誤的地方，卻又不知道從哪裡改起，因為編譯器所提示的錯誤都是英文加上只給語法錯誤。

C 語言在 70 年代初期就已經有了迅速的演進，而 1972 年貝爾實驗室為了重寫 UNIX 作業系統，開始研發 C 語言，也因此 C 語言有了大幅的改進，其中最重要的就是前置處理器的發明。在 73 年的夏季，現代版的 C 語言可說是雛型大致完備了。1973 年至 1980 年期間，C 語言又增加了 unsigned、long 等等數種資料型態。而這時貝爾實驗室也嘗試著將一些程式轉移到其它的作業平台。比較重要的是西元 1977 年，Steve Johnson、Ken Thompsen 與 Dennis M.Ritchie 開始把 UNIX 移到 Interdata 8/32 的機器上。當 Interdata 8/32 移植成功之後，Tom London 與 JohnReiser 也把 UNIX 移到 DEC 的 VAX11/780 電腦。而 VAX 11/780 電腦的流行遠遠超過 Interdata 8/32，於是 UNIX 與 C 語言便迅速地傳播擴展。而 C 語言更是在 80 年代被廣泛地使用，編譯器幾乎可以在任何一種機器與作業系統上看見，尤其是在個人電腦大為流行之後，更是成為發展 PC 應用程式的最佳工具。C 語言不僅移植能力高且其結構嚴謹，執行速度也非常的快，更具有一般高階語言親合力強的基本特性。C 的兩個著名標準：K＆R 與 ANSIC，更是促使了 C 的移植性大大地提高。我們在寫程式的時候只要遵循其規則，避免使用到標準程式庫以外的程式庫，而且儘量符合 ANSIC 的標準，那麼這些程式碼想要轉到其他的編譯器甚至是作業系統就更不是問題了，而 C 可應用的領域也就越大了。

C 語言有幾項特點：

1. C 語言函式的區段架構，讓原本很大的程式不斷的向下分解成一些較簡單的片段，讓每一個片段具有一個小功能，而在測試時也可分段來測試，不需每次都要重新全部執行再行測試，減少了很多時間，也將程式變的較為簡化。
2. C 語言提供數種如 while、for、switch 等的結構化敘述來取代 goto 敘述，可有效的避免類似過去在組合語言等程式語言中，由於 goto 敘述不當使用所造成的錯誤。
3. 在其函式庫中每個函數，甚至是自己寫的函式，都可具有其自己的功能，獨立存在於程式之內，成為 C 語言的延伸，而這些函式可透過呼叫重複的使用，如此一來不僅減少了程式發展的時間也使程式碼不至於過於龐大。

4. 將編輯系統 (editor)、編譯系統 (compiler) 及偵錯系統 (debugger) 整合在一起，隨時可以呼叫使用，這種整合式的工作環境我們稱為整合式發展環境 (IDE：Integrated Development Environment)，是近代新式語言系統所採行的方法。
5. 需要長時間的訓練，也就是需要一些比較專業的技巧。

寫程式時常遇到的困難。一)編譯器會檢查程式是正確並且記錄錯誤的訊息但是一些學生無法理解錯誤是什麼意思，所以他們無法使用這些資訊去偵錯，他們覺得錯誤訊息是英文的很難理解。二)多標準函式庫程序學生常常會誤解他們，網路上有線上人工"man"可以隨時使用，但是學生常不去使用它，因為學生覺得裡面的敘述太困難。三)在寫程式時，如何替變數或函式命名是很重要的，但是國小、國中生的詞彙通常都很少，所以就算學生知道這命名的重要性，這對他們來說還是很困難的。四)型別的錯誤在編譯器上會造成許多的錯誤，所以型別如何宣告也是必須要有經驗的，但是如何去命名也能幫助減少錯誤。五)因為有些錯誤是很近似的，所以很多學生都會重複的犯同樣的錯誤，所以學生應該要更小心地去理解這些錯誤，知道這些錯誤的差別是在哪。

在我們的系統中，編譯器和資料庫佔了舉足輕重的地位，圖 1 是編譯器架構圖。Scanner: 會逐個逐個字元讀進來並且在“適當時候”把字元合成一組 Token 供後邊的 Parser 做其他處理工作。Parser: 有了 Scanner 幫我們把字元合併成 Token 了，再來就是要把 Token 組合成有意思的“句子”了，這一步稱為語法分析，而負責做這項的作的程式我們稱之為語法分析器(Parser)。Immediate code: 是從語法樹中，以一個節點(node)為基本單位，從最底層的節點依序往上，拆解成一個個最基本的運算式，而每一個節點也會賦予一個暫時性的符號。Code Generation:以 C 語言為例，這裡就是將最佳化後的中間碼，搭配微處理器的暫存器，逐一轉換成組合語言。Assembler: 使用組合語言編寫的原始碼，然後通過相應的組譯程式將它們轉換成可執行的機器碼。這一過程被稱為組譯過程。Linker: 連結器的用途是將相關的程式目的碼(由編譯器所產生之 OBJ 檔)與庫存函數連結起來。

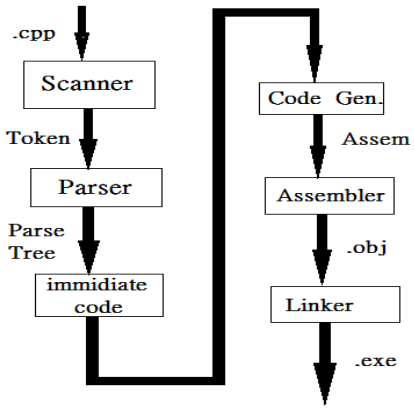

圖 1. 編譯器架構圖

電腦程式就像所有人類的產物一樣，通常都無法盡善盡美。電腦程式碼可能包含不同類型的錯誤。錯誤可能是語法錯誤、語意(semantic)錯誤或邏輯錯誤。最明顯的錯誤類型為語法錯誤，它發生於程式碼編寫方式不符合語言規則時。語法錯誤幾乎都可由編譯器(compiler)或解譯器測出，其將顯示一個通知您發生問題的錯誤訊息。語意錯誤則為較難捉摸的錯誤類型。語意錯誤會發生在程式碼的語法正確，但是其語意或含意並非您所意指之時。由於語法結構符合語言規則，所以無法由編譯器或解譯器找出語意錯誤。編譯器和解譯器的關切重點只在編寫的程式碼之結構，而非其含意。語意錯誤可能導致程

式不正常結束，並可能顯示或不顯示任何錯誤訊息。以口語說明就是，語意錯誤可能使程式損毀或當掉。不過，並非所有的語意錯誤都會用這種明顯的方式來表示。一個程式可能會在語意錯誤出現之後繼續執行，但是其內部狀態將與原來所想的大不相同。變數可能不包含正確的資料，或者該程式會一直在不希望執行的路徑上發生錯誤。最後產生的結果將是不正確的輸出。這些錯誤稱為邏輯錯誤，因為雖然程式並未損毀，所執行的邏輯卻是錯誤的。偵測通常是邏輯錯誤的唯一方法，是以手動或自動方式測試您的程式，並確認輸出是您所預期的測試應為軟體開發過程中不可或缺的一環。可惜的是，雖然測試可告訴您程式的輸出不正確，但卻無法留下讓您瞭解到底程式碼的哪個部分導致問題的線索。偵錯通常但並不一定會使用偵錯工具，這種功能強大的工具可讓您觀察程式的 Run-Time 行為，並找出語意錯誤的位置。您也可以使用程式語言和其相關程式庫內建的特定偵錯功能。許多程式設計人員在嘗試隔開問題時，會在程式碼內加入輸出函式的呼叫，例如 printf 或 MsgBox，這是他們在一開始接觸到偵錯工作時的做法。這是一種相當合理的偵錯技巧，不過當找到並修復問題之後，必須回頭檢查所有程式碼，並移除所有額外的函式呼叫 (Function Call)。可能偶爾也會發現加入新程式碼 (即使僅是一個 printf 或 MsgBox 呼叫)，亦會改變要偵錯的程式碼之行為。有時可以透過偵錯工具，來檢查程式內的變數內容，而不需要插入額外的呼叫來輸出數值。也可在程式碼內插入中斷點，以便在有興趣的位置停止執行。您可以在程式停止時 (在中斷模式下)，使用相關功能來檢查區域變數 (Local Variable) 和其他相關的資料，例如 [監看式] 視窗、[快速監看式] 對話方塊和 [記憶體] 視窗。如需詳細資訊，請參閱監看式視窗、快速監看式對話方塊或記憶體視窗。您不僅可在中斷模式中檢視內容，還可視需要編輯或變更其內容。在大多數情況下，您可在用來撰寫與編輯程式碼的 Visual Studio 來源視窗內設定中斷點。有時候，您可選擇在偵錯工具的[反組譯碼]視窗內設定中斷點。這個[反組譯碼]視窗會顯示出由原始程式碼所建立的指示。如需詳細資訊，請參閱反組譯碼視窗。與 printf 或 MsgBox 不同的是，設定中斷點並不會在原始程式碼內加入額外的函式呼叫。因此，設定中斷點不太可能變更您要偵錯的程式之行為。

對程式設計來說，除錯有許多意義，多數開發者學習程式設計的第一關卡，就是跟編譯器或者直譯器搏鬥，根據編譯或直譯不通過的訊息找出問題，直到寫出型態正確且符合語法的程式碼。許多時候，除錯的難度並不在於錯誤本身多困難，而在於難以找到錯誤的所在地，這是除錯與測試的不同點，測試可以驗證某個函式呼叫結果有錯誤，然而，這錯誤是在函式本身的哪個地方產生呢？或者是函式又呼叫了某個子函式而產生？除錯常常是設計師頭痛的來源。每個程式幾乎都不免有錯，不論是生手或老手的作品都一樣。我們在設計程式之初總會有美麗的憧憬：當我作好規劃，然後編碼，再修改一些『小錯誤』，這個程式就完成了！事實不然，或許你寫程式的速度的確很快，但你永遠也無法知道你將會花多少時間和精力在除錯上，這真是一個可悲的事實！另一方面，『除錯』這件工作即使是在科學昌明的今日，也缺乏一套好方法，和一些好工具來協助我們除錯，雖然新的除錯工具不斷的在出現，但除錯依舊還是一件令人沮喪的工作。

錯誤是可以分類的，特別是可以用出現頻率來分別，一直發生，常常發生，偶而發生，甚至發生過一次後就再也不發生的錯誤。而搜索錯誤的方式也有一些常見的方式：暴力法、往前搜尋法、原因消除法。暴力法是最常見、最沒效率的方法，除錯的人總是一步一步的從頭追蹤程式，把記憶體中的變數甚至檔案的內容都印出來，或是在程式中很多地方顯示執行時的情況.....當然這種方法可以找到錯誤，但需要很長的時間和很多精力，不過我們還是會運用這個方法，特別是在一切都絕望的時候。往前搜尋法或許是較好的方案。從錯誤發生的地方往前找看看到底那裡不對勁。在程式較小的場合或許管

用，但程式太大了就困難了。最科學的當屬原因消去法。我們有系統的去分析一個錯誤發生的原因，作一些可能的假設。就像是汽車發不動了，你不會急忙的把車子支解，而是會想想看那些零件的故障會導致車輛無法發動，例如火星塞、分電盤.....。以上提的三種除錯的方法都有適當的軟體工具來配合，例如單步追蹤器，或中斷點除錯.....等。不可否認的，除錯的工作是一件令人痛苦而不耐的工作。

之前曾有藉由使用代理技術提出了一個支援 C 語言程式設計在 UNIX 上進行編譯的系統。此系統監視來自編譯器的錯誤消息。它會分析並重寫獲取的消息，並在網路上報告給學生和老師。老師可以通過查找錯誤的類別來改善教學方式。學生自定義這些訊息，就可以在每個級別裡使用較合適的，此外，這些訊息通過每個通訊線路來送給其他學生代理，因此，許多學生能夠使用這些有用訊息。此系統可以從代理的集合裡選擇最合適的訊息，但是卻還沒提供一個完整的介面供學生去使用。

GNU 偵錯器(GNU Debugger，縮寫：GDB)，是 GNU 軟體系統中的標準偵錯器，此外 GDB 也是個具有移攜性的偵錯器，經過移攜需求的調修與重新編譯，如今許多的類 UNIX 作業系統上都可以使用 GDB，而現有 GDB 所能支援除錯的程式語言有 C、C++、Pascal 以及 FORTRAN。GDB 具備各種偵錯功效，能針對電腦程式的執行進行追蹤與警告，使用 GDB 的除錯人員可以監督及修改程式的內部變數值，甚至監督與修改獨立於主程式運作外，以獨立個體型態呼用(呼叫使用)的函式。GDB 能為多種不同處理器架構上執行的軟體進行偵錯，這些處理器架構包括：DEC/COMPAQ/HP 的 Alpha、ARM 的 ARM、Hitachi 的 H8/300、IBM 的 System/370、System 390、Intel 的 X86 及 X86-64、IA-64 "Itanium"、Motorola 68000、MIPS 的 MIPS、HP 的 PA-RISC、AIM 聯盟的 PowerPC、Hitachi/Renesas/ST 的 SuperH、Sun 的 SPARC、DEC/COMPAQ/HP 的 VAX。此外一些比較少人知的處理器也一樣受 GDB 支援，包括：AMD 的 A29K、ARC 的 ARC、Atmel 的 AVR、Axis Communications 的 ETRAX CRIS、D10V、D30V、Fujitsu 的 FR-30、FR-V、Intel 的 i960、Renesas 的 M32R、Motorola 的 68HC11、Motorola 的 88000、Freescale 的 MCORE、MN10200、MN10300、NS32K、Stormy16、V850、以及 Zilog 的 Z8000，此外更後續版本的 GDB 只會增加更多的處理器支援而不會短少。GDB 的內部已具備了依據各種支援不同處理器的指令集所編譯成的模擬推演程式(Simulator)，就連大眾少知少用的 M32R、V850 等架構的處理器也都具備。GDB 的一大特點是：除了具備傳統的本機端、近端偵錯外，也可透過接線、網路的通訊方式進行遠端性的遙控偵錯。GDB 具有一種「遠端，Remote」模式，此種模式多半是在為嵌入式系統進行偵錯時使用，遠端操作指的是：GDB 在一部機器內執行，而要進行偵錯的程式是在另一部機器上執行，接著欲偵錯的機器上會再加裝一個名為「Stub」的小程式，該程式能夠與另一端的 GDB 程式溝通，溝通的路徑可以是兩部機器間的串接式接線（Serial Cable），也可以是支援 TCP/IP 協定傳輸的各種網路，在 TCP/IP 網路及協定上再加搭傳輸 GDB 專有的除錯操作用通訊協定，如此便能進行遠端偵錯。不僅 GDB 有遠端模式，KGDB 也同樣具有遠端模式，KGDB 主要是為執行中的 Linux 核心進行偵錯，而 GDB 則是主要是用在原代碼的層次。運用 KGDB，負責核心程式的程式設計師可以將核心以近似於應用程式的除錯方式來偵錯，包括為核心程式碼設置中斷點（breakpoint）、讓核心程式以步階方式逐行執行以及觀看變數值等。在某些架構的處理器中，會以硬體方式提供一些偵錯功用的暫存器，以及可以設定觀察點(Watchpoint)，觀察點的功用是：當程式設計師指定的記憶體位址被執行到或存取到時，觀察點即會去觸發、觸動一個中斷點。對此 KGDB 可以安裝在一部傳統機器上，並透過遠端模式使用另一部受偵錯機器上的硬體偵錯功效，同樣的兩部機器可用各種方式進行溝通，如串接式接線、乙太網路等，尤其在 FreeBSD 作

業系統上還允許使用 FireWire 接線，並用直接記憶體存取（Direct Memory Access，DMA）的功效來協助遠端偵錯。GDB 的 GUI 化前端程式：Insight（過去稱 GDBtk），圖中為「堆疊瀏覽器」部分。GDB 運用上最明顯的限制是在「使用者介面」的部分，預設只有命令列介面(CLI)可用，而不具備較能親合上手、直覺操作的圖形化使用者介面(GUI)，不過此一弱處也已經有幾個前端程式為其補強，例如 DDD、GDBtk／Insight 以及 Emacs 中的「GUD 模式」等，有了這些補強後，GDB 在功效使用的便利性上就能夠與「整合發展環境中的偵錯功效使用」相接近。在我們的系統中，我們希望能建立如 GDB 的 C 語言編譯平台且改善編譯環境調適，分析錯誤，重寫獲取的訊息，並印出在平台上給學生看

為了創造題庫我們需要建立資料庫。一個資料庫系統主要組成包括：資料、硬體、軟體及使用者。資料：係由許多相關聯的檔案所整合而成，資料庫內的個別資料，可由多人共 同使用，或是讓不同使用者在同一時間使用同一筆資料。硬體：即磁碟、硬碟等輔助儲存設備，或一切的週邊設備。軟體：資料庫管理系統(Data Base Management System, DBMS)。使用者：一般使用者、程式設計師及資料庫管理師(DBA)。本計畫將配合題庫的練習，讓學生不會剛開始就對寫程式感到無趣，就好比學數學一樣，老師如何將題目簡單化，並用最容易讓人懂得解題方式來讓學生一目了然，讓學生會對數學產生興趣。

2. 系統架構

本系統平台不只希望可以幫學生把錯誤做適當的翻譯，還希望可以幫學生把錯誤自動更正好，只要學生將整段錯誤程式碼貼到平台上，編譯之後，會將正確程式碼呈現出來，例如原本他在宣告一個變數時，忘記初始化，那麼更正後會自動幫他給一個初值，最後，還會在平台上建立一個題庫，這個題庫裡會將學生常犯的錯誤，例如型別宣告 stack overflow 等這些放上去，裡頭的題目會是 C 編譯器上顯示的錯誤碼，答案則是放適當翻譯過後的錯誤資訊，學生可以藉由練習題庫的方式，去對這些編譯器上顯示的錯誤能更加熟悉更快速地去找出錯誤發生在哪個地方。圖 2 是平台流程圖。

本系統平台可以在網站上執行這些功能，所以只要在 windows 上就能執行，而目前國外所做的是必須在 UNIX 上執行，但是我們所針對的是國小、國中生，如果還要他們去自己安裝 UNIX 系統，那也等於是為難他們，或許他們本來想使用，但是想到還必須去安裝 UNIX，就不想去使用他們開發的這套系統了，而且，Unix 是科學計算、大型電腦、超級電腦等所用作業系統的主流。所以，在我們這種個人電腦或許沒辦法普及化。但是如果是在平台上，操作起來就方便許多，因為現在的國小、國中生幾乎天天都會去瀏覽一些平台，所以他們使用起來一定會比在 UNIX 上簡單許多，並且還提供了題庫區，因為大部分觀念還是必須要靠題目來加強，否則遇到問題就去平台查一次這錯誤碼是什麼意思可能程式功力也進步不了。

我們會寫一個編譯器來，因為要寫的是 C 語言的 Compiler，所以需要先寫 "語義解析"，用來分析 C 語言的語法以及結構，接下來可以把分析好的結果轉換成組合語言，然後再寫個 Assembler 把組語轉成機器碼(OP code)，當然，中間還需要寫個 Linker 來鏈結 Assembler 產出的 object code 以產出最終可以執行的碼。網路上可以查詢到一些線上編譯器，但是要做到幫別人改正錯誤，是最困難的地方，是因為語法錯誤有千奇百種，特別最難是邏輯上的錯誤，或許我們能做的就只能幫他們將程序中的變量列印到螢幕，以定位錯誤所在。雖然這個辦法不是所有情況下都適用，比如在調用了錯誤的子例程時就行不通，但是如果程序由於計算結果有誤而出錯，這種方法是最能幫助他們找出問題

的所在。題庫的構圖主要是分為選擇題和是非題兩部分，題目主要是放一些初學者常犯的錯誤進去，選項就放我們將那些錯誤翻譯之後的結果。

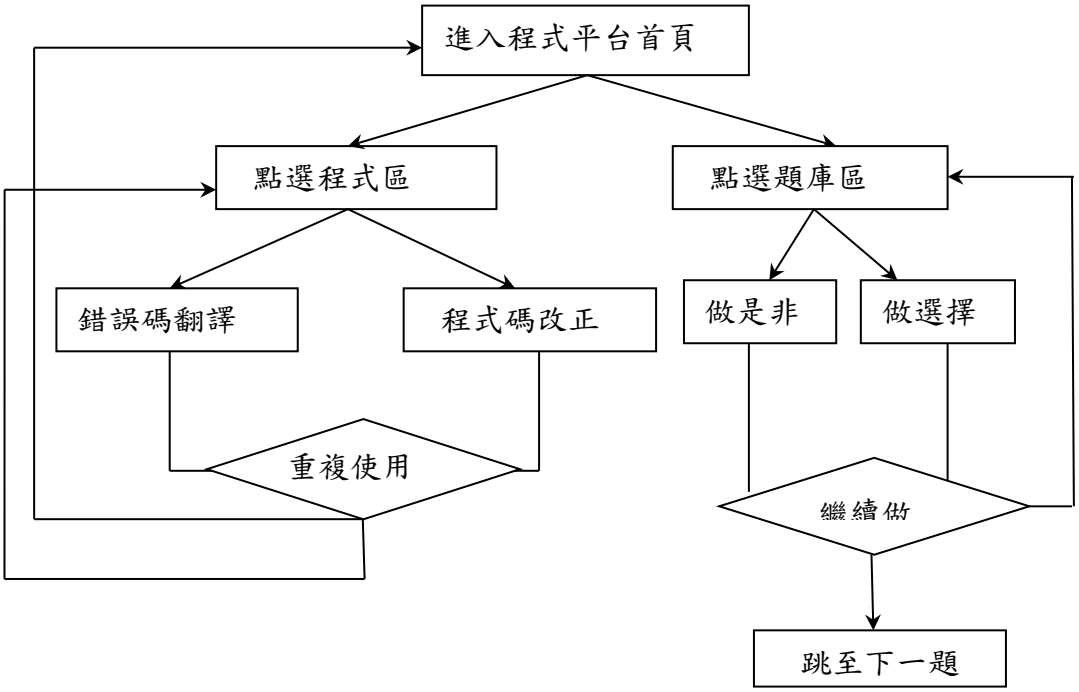

圖 2. 平台流程圖

圖3是本中小學生C語言學習平台方塊圖，包含一身份辨認登入端，用以識別使用者身份，以決定進入C語言平台中的哪一單元，該身份辨認登入端是現已廣泛使用的網站登入介面，可以判別使用者身份，決定是否有登入的權限，該身份辨認登入端可經由一網頁設計技術搭配資料庫，產生使用者身份資料IDdata、密碼資料PWdata與一時間資料Timedata。C語言學習平台，其輸入端連接身份辨認登入端，以接收使用者身份相關資料，該C語言學習平台，包含一程式學習單元如圖4所示，此單元中有一編輯單元，編輯單元是一C語言的開發環境，提供程式的開發者一個完整的環境，輕鬆地編輯C語言的程式碼PCcode，輸入端連接身份辨認登入端，當確認身分為學生時，提供學生編輯C語言程式原始碼，以輸出程式虛擬碼；編譯連結單元整合文字編輯器、編譯器(compiler)、除錯工具(debugger)及建置工具等等，語法錯誤就好像一個句子漏了名詞和動詞等，例如打漏了等如符號，括號數目不平衡等，那麼編譯器就懂不得程式。而語義錯誤就好像串錯了字，就算編譯器懂得看，也不懂得編譯程式。如果有上述錯誤，編譯器會通知你，而且停止編譯過程，這時你要修正程式內的錯誤，修改後重新開始編譯器的工作，當沒有任何錯誤後，該編譯連結單元會把每個語言句子轉成「機械語言」(MachineCode)的「目的碼」(ObjectCode)，產生另一個檔案"file.obj"，連結(Link)過程，會把有關的程式和程式庫所產生出來的*.obj，轉成可以在電腦上執行的方式，產生另一個檔"file.exe"，這個檔案就可以執行了。。編譯連結單元輸入端連接編輯單元，將程式虛擬碼進行編譯連結，以產生一目的檔，如目的檔成功產出，即可輸出一執行檔至翻譯批改端，若失敗，則將錯誤訊息送至翻譯批改端譯成淺顯易懂的中文，供中小學生了解改正。

題庫單元顯示在圖5，包含一練習單元，其輸入端連接身份辨認登入端，以接收身份資料與時間資料，當確認身分為學生時，提供學生選擇與是非兩種題型測驗，以產生答案資料；以及一出題單元，其輸入端連接身份辨認登入端，當確認身分為教師時進入，

包含：一新增刪除修改單元，當登入身分為教師時，教師可在此單元對題目作新增、刪除及修改等動作；以及一答案單元，當登入身分為教師時，教師可在此單元對所有題目提供答案，作為批改之用；以及一翻譯批改端，連接C語言學習平台，以輸出翻譯的錯誤訊息、執行或批改結果。新增刪除修改單元透過網頁介面操作，可隨時隨地使用各種裝置進行教學。答案單元使用自動化雲端裁判系統判別學生作答結果，無需人工執行程式。

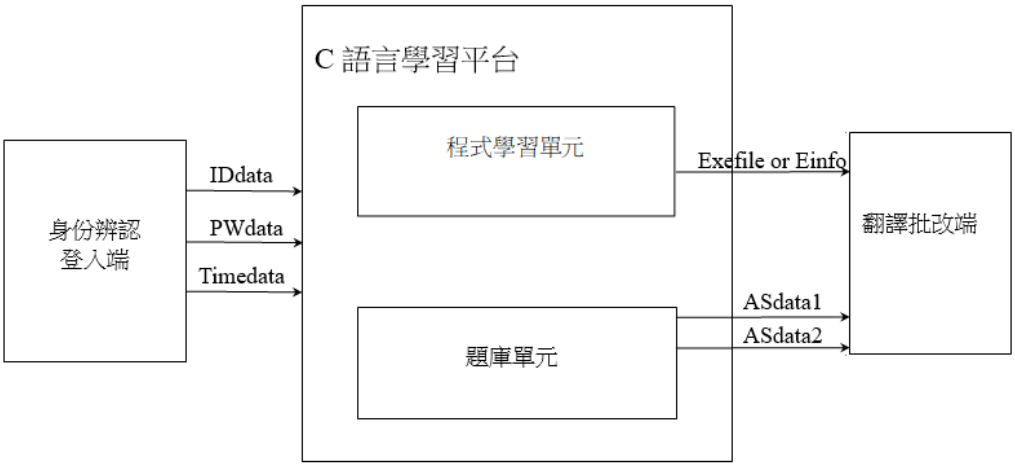

圖3. 平台方塊圖

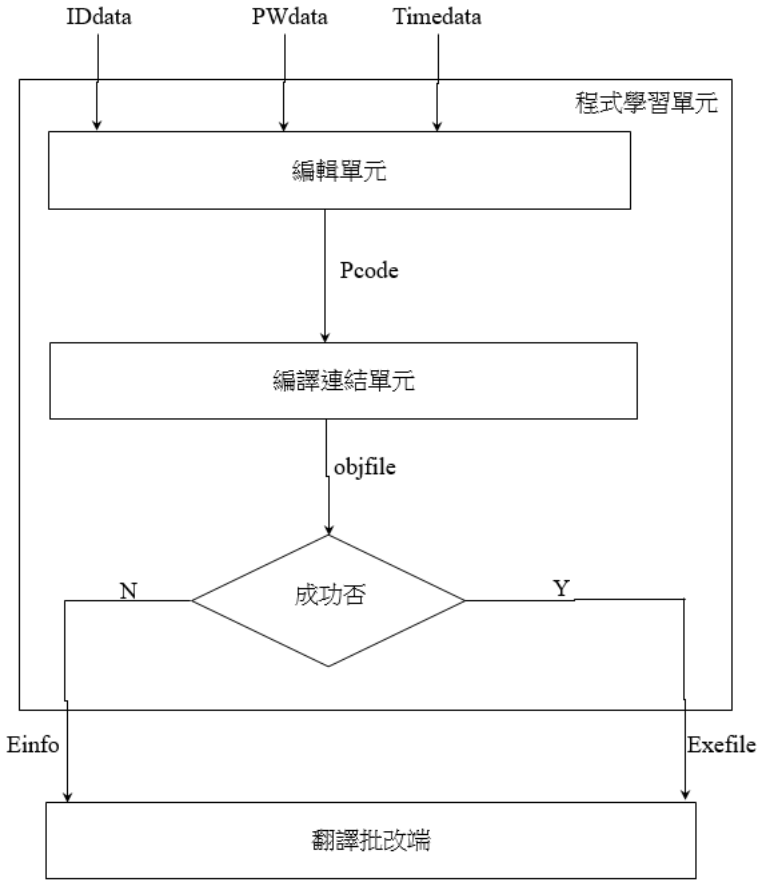

圖4 程式學習單元

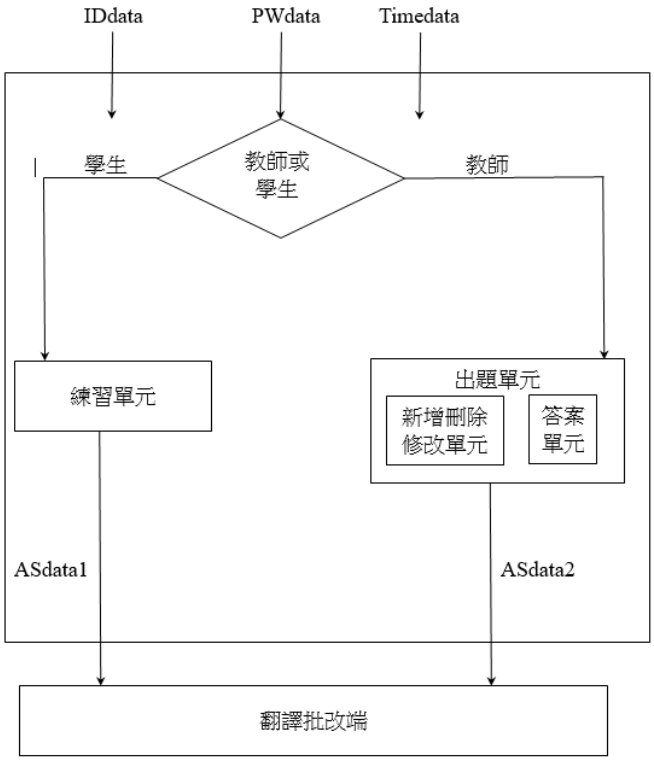

圖5 題庫單元

3. 實務開發

本系統實現的介面如圖 6~圖 10 所示。圖 6 為系統入口，圖 7 為編譯及題庫入口系統，圖 8 為題庫系統，圖 9 顯示討論區，後臺則顯示在圖 10。

圖 6 系統入口

圖 7 編譯及題庫入口系統

圖 8 題庫系統

圖 9 討論區

圖 10 後臺系統

4. 參考文獻

[1] 何榮桂(2001)：他山之石可以攻錯~ 亞太地區(臺、港、新、日、韓) 資訊教育的發展與前瞻，資訊與 教育，81，1-6。

[2] 吳正己(2000)：高中電腦教科書編撰建 議 ， 國 立 編 譯 館 通 訊 13(2) ， 11-17。

[3] Cathy Bishop-Clark, Jill Courte, Elizabeth V.Howard (2006). Programming in Pairs with Alice to Improve Confidence, Enjoyment, and Achoievement. Baywood Publishing Co., Inc. 213-228.

[4] Elizabeth V.Howard, Donna Evans, Jill Courte, Cathy Bishop-Clark (2006). A Qualitative Look at Alice and Pair-Programming. Proc ISECON 2006, v23, 1-8.

[5] Schulte, C., Magenheim, J., Niere J., & Schäfer W. (2003). Thinking in Objects and their Collaboration: Introducing Object-Oriented Technology. Computer Science Education 2003, vol. 13. No. 4. pp. 269-288

[6] Stephen Copper,Wanda Dann, Randy Pausch (2000). Alice: A 3-D Tool for Introductory Programming Concepts. JCSC, the Consortium for Computing in Small Colleges, 108-117.