國立屏東大學 資訊工程學系 C++程式設計入門教材

1. 電腦系統與程式語言

2016年3月，Google DeepMind公司的AlphaGo電腦圍棋軟體以四勝一負擊敗了南韓棋王李世乭。這場對弈標示了人工智慧(Artificial Intelligence，AI)的發展邁向了新的里程碑，人們開始注意到電腦也可以在腦力上戰勝人類，甚至開始擔心未來電腦會不會完全取代人類？在許多科幻小說與電影中，電腦系統總是被描繪為在未來將具有超越人類能力並能自主思考的AI，這些過份聰明的電腦系統，通常並不會乖乖地為人類服務，而是更聰明地發射核彈將人類消滅殆盡[1)]，或是將人類豢養起來做為生物電池以提供電腦系統源源不絕的能源[2)]。總之，有了太過聰明的電腦似乎不是什麼好事，就連史蒂芬．霍金(Stephen Hawking)都曾說過：

> The development of full artificial intelligence could spell the end of the human race![3)]
>
> 翻譯吐司→(人工智慧發展到了極致，可能會導致人類的滅亡！)

不過請讀者們暫時不用擔心，從在特定領域以智能打敗人類，到能夠真正發展出擁有獨立思考能力的電腦系統，仍然還有很長的一段路要走。你可以放心地過好每一天，至少到目前為止，電腦系統仍然是聽命於人類，乖乖地執行我們所交付的工作。當然，做為資訊人的一份子，或許你將來也會為實現具備AI的超級電腦而努力呢！

電腦系統可以簡單地區分為硬體(Hardware)與軟體(Software)，例如個人電腦、智慧型行動電話、平板電腦、智慧型手錶等看得到、摸得到的實體裝置就是硬體；我們可以在這些硬體上執行各式不同應用目的的軟體，以滿足我們各種不同的需求 — 例如我們可以開啟文書編輯軟體來編輯一份文件、開啟瀏覽器來查詢資料、玩游戲軟體以消磨時間、或是利用軟體來進行多媒體影音作品的創作等。軟體是由資料以及一個或多個程式(Program)所組成，可以在電腦系統上執行以完成特定的工作。簡單的軟體可能僅由一個程式組成，而複雜的軟體則可能由多個程式組成，其中每個程式負責特定的一部份功能。對於硬體而言，一個程式是一些機器碼(Machine Code)[4)]的集合，將這些機器碼加以執行，就可以解決或滿足使用者特定的問題或需求。我們在前面提到的「過份聰明的電腦系統」，就是具備先進AI的電腦系統，並且會「自已」產生各種可以用來消滅或奴役人類的程式。

在目前的現實生活中，電腦系統所執行的各種程式仍是由我們人類所開發的 — 我們必須遵循程式語言(Programming Language)的語法規則，將所要執行的各種功能撰寫為原始碼(Source Code)，再經過編譯(Compile)將其轉換為可在電腦系統中執行的機器碼後，才能交付給電腦系統執行。本章將針對電腦系統(Computer Systems)的基礎知識加以說明，其中包含當代電腦系統的基礎 — 儲存程式型電腦架構(Stored-Program Computer Architecture)，以及電腦系統的硬體、軟體、程式語言，以及本書的主角 — C++語言，逐一加以簡介。

1.1 儲存程式型電腦

現今的電腦系統都是以著名數學家約翰．馮紐曼(John von Neumann)於1945年所提出的儲存程式型電腦架構(Stored-Program Computer Architecture)[5)6)7)]為基礎，一般又被稱為馮紐曼模型(von Neumann Model)[8)9)]，其主要概念是我們可以將工作事先撰寫為程式(Program)並加以儲存，等到需要時再將其載入

電腦系統加以執行即可完成工作。

隨著資訊科技的發展，電腦系統朝向行動化與普及化發展，有愈來愈多不同形式的電腦系統問世，除了傳統的桌上型個人電腦、筆記型電腦、各式伺服器、大型主機外，還有許多新穎的可攜式系統、嵌入式系統正融入我們的生活，包括智慧型行動電話、平板電腦、智慧型手錶、手持式/車用衛星導航系統、甚至連許多家電或生活用品內都可看到電腦系統的蹤影。雖然這些電腦系統的功能、形式皆不盡相同，但都同樣都是以儲存程式型電腦架構為基礎。

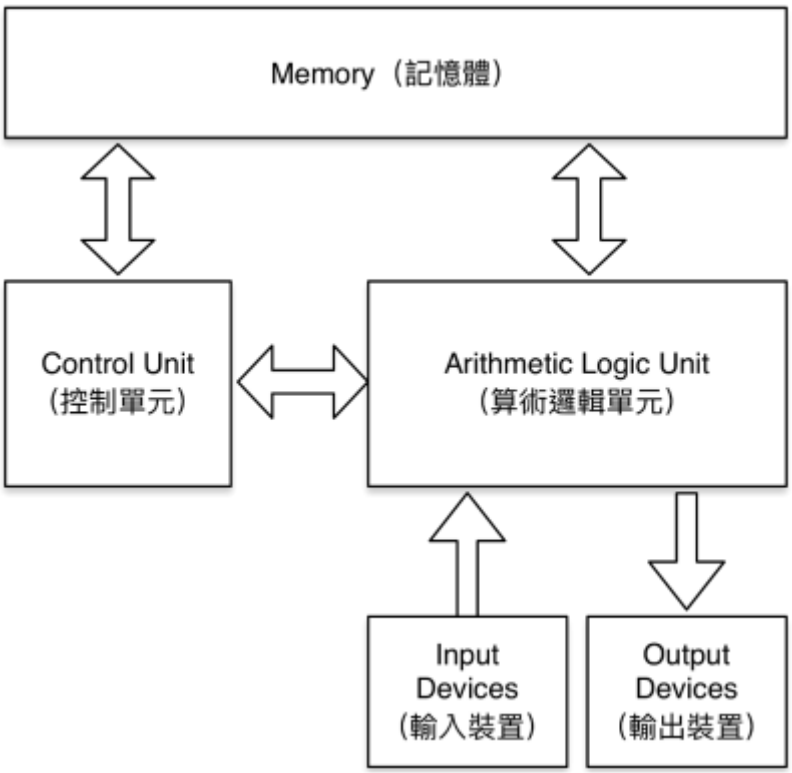

Fig. 1: 儲存程式型電腦架構(馮紐曼模型)

在儲存程式型電腦架構中，電腦系統包含記憶體(Memory)、算術邏輯單元(Arithmetic Logic Unit、ALU)、控制單元(Control Unit)與輸入/輸出裝置(Input/Output Device)等四大單元，如figure 1所示。其中記憶體是用以儲存資料與程式的空間，算術邏輯單元負責執行算術與邏輯運算，例如兩個數字的加減乘除或比較其大小等工作。控制單元則是用以控制記憶體、算術邏輯單元及輸入/輸出裝置的運作，通常與算術邏輯單元合稱為中央處理器(Central Processing Unit、CPU)。至於輸入/輸出裝置則負責接收資料與程式(例如透過鍵盤與滑鼠來將資料輸入電腦系統，或是從磁碟來讀取資料或程式)，並且將運算後的結果傳送出去(例如螢幕、印表機或是將結果儲存在磁碟內)。

在儲存程式型電腦架構內的記憶體除了可儲存資料外，也被用來儲存程式，當我們需要執行不同的程式時，只要將其載入到記憶體中即可由算術邏輯單元加以執行。因此，我們可以視需要載入不同的程式到電腦系統裡加以執行，來完成不同的工作。透過此一概念的延伸，我們可以將電腦系統區分為硬體(Hardware)與軟體(Software)兩部份，兩者必須搭配運作才能完成特定的工作，單有硬體而無軟體是無法運作的，反之亦然。

1.2 電腦硬體

所謂的電腦硬體(Computer Hardware)係指電腦系統內實體的部份。雖然現今的電腦系統存在許多不同的形式，但絕大部份都仍符合儲存程式型電腦架構，其中的硬體元件可概分為記憶體(Memory)、算術邏輯單元(Arithmetic Logic Unit)、控制單元(Control Unit)與輸入/輸出裝置(Input/Output Devices)等四大單元。通常記憶體又可區分為主記憶體(Main Memory)與次記憶體(Secondary Memory)。其中次記憶體包含了儲

存裝置，例如硬碟、軟碟與光碟機等，更常被稱為次儲存體(Secondary Storage)。算術邏輯單元則與控制單元合稱為中央處理器(Central Processing Unit，CPU)，亦可簡稱為處理器(Processor)。至於輸入/輸出裝置則包含鍵盤、滑鼠、掃瞄器等輸入裝置與螢幕、印表機等輸出裝置。請參考figure 2，一個典型的電腦系統要執行一個程式時，首先會將儲存在次記憶體的程式載入到主記憶體中，接著透過輸入裝置讓使用者輸入程式所需的資料並同樣儲存到主記憶體裡，再由CPU執行程式並將執行結果儲存回主記憶體或是透過輸出裝置將結果呈現給使用者。本小節將就這些相關的硬體做一簡介。

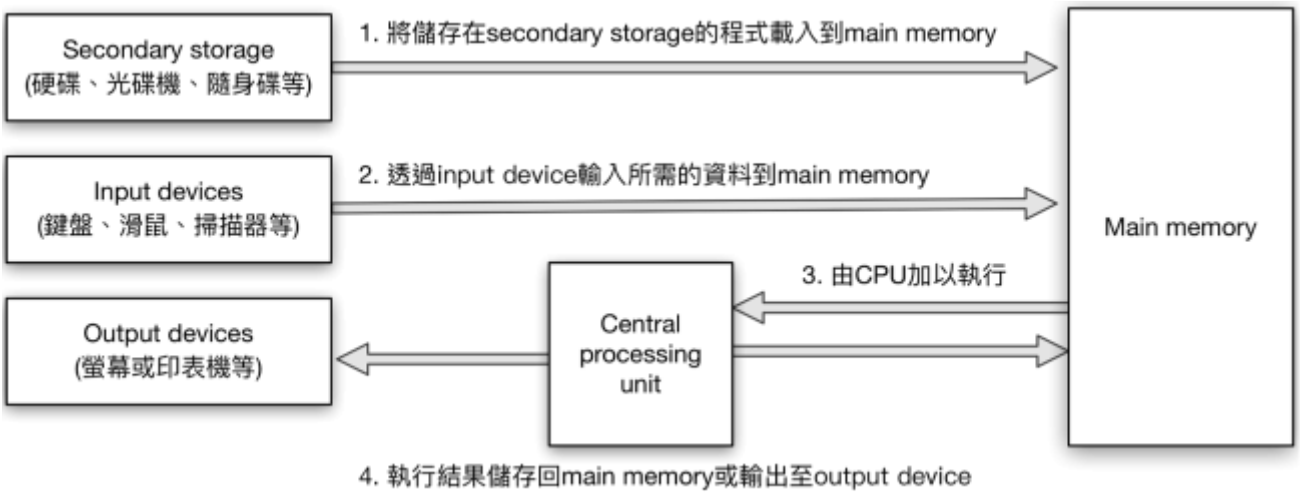

Fig. 2: 典型的儲存程式型電腦架構硬體元件與程式執行流程

1.2.1 主記憶體

主記憶體(Main Memory)是用以儲存程式、資料與運算結果的硬體元件。一般又可依其特性分為隨機存取記憶體(Random Access Memory，RAM)與唯讀記憶體(Read-Only Memory，ROM)兩類。RAM可以用來儲存要被執行的程式，並在執行時用以存放相關的數值、文字、甚至是圖片、影像等資料。但是RAM僅能暫時性的儲存資料，因為它是所謂的揮發性記憶體(Volatile Memory)。當電腦系統關閉電源或重新啟動後，保存在RAM裡的資料都將不復存在。相對的，ROM則可長久地保有資料，稱為非揮發性記憶體(Nonvolatile Memory)。唯讀記憶體就如同其名稱一樣，僅能讓我們讀取資料，並不能寫入新的資料，其所儲存的資料與程式是由硬體廠商事先加以燒錄，通常是負責電腦系統啟動時所要執行的程式及相關資料。本書所指的主記憶體是指RAM而非ROM。因為我們只有對RAM才能進行讀取與寫入的操作。由於儲存程式型電腦架構必須先將資料與程式載入到RAM後才能加以執行，所以RAM是十分重要的硬體元件。

資料在記憶體內是以二進位的方式儲存，通常由一系列大小相同的記憶單元(Memory Cell)所構成。電腦系統對記憶體的操作就是對記憶體單元進行「存」或「取」的操作：

- 存：儲存資料到記憶單元內(其原本的內容將會被新的資料所取代)，或稱為寫入(Write)資料。
- 取：取回在記憶單元裡的資料(其原本的內容將不會被改變)，或稱為讀取(Read)資料。

事實上，記憶體單元並不是最小的儲存單位，在記憶體裡最小的儲存單位稱為位元(Bit，或簡寫為b)。也就是用以儲存一個0或一個1的空間，而在電腦系統裡的資料(不論是數值、文字符號或程式指令)都是由一系列特定樣式的0與1所構成。通常我們將連續的8個位元稱為一個位元組(Byte，或簡寫為B)，可用以表達基本的數值或文字符號，例如以 ” 00010100 ” 表示一個英文字母，A，或是一個數值「20」。table 1列示了常用的記憶體空間單位。

記憶體單位	縮寫	空間大小
Bit 位元	b	僅可存放一個0或1的最小記憶體單位

記憶體單位	縮寫	空間大小
Byte 位元組	B	1 byte = 8 bits = 2^3 bits
Kilobyte 仟位元組	KB	1 KB = 1024 bytes = 2^{10} bytes
Megabyte 百萬位元組	MB	1 MB = 1024 KB = 2^{20} bytes
Gigabyte 十億位元組	GB	1 GB = 1024 MB = 2^{30} bytes
Terabyte 兆位元組	TB	1 TB = 1024 GB = 2^{40} bytes
Petabyte仟兆位元組	PB	1 PB = 1024 TB = 2^{50} bytes
Exabyte百京位元組	EB	1 EB = 1024 PB = 2^{60} bytes
Zettabyte十垓位元組	ZB	1 ZB = 1024 EB = 2^{70} bytes
Yottabyte秭位元組	YB	1 YB = 1024 ZB = 2^{80} bytes

Tab. 1: 常見的記憶體單位

現今電腦系統內的記憶體通常都具有數以百萬計的記憶體單元，為了要能夠存取在不同記憶體單元裡的資料或程式指令，電腦系統首先必須為每個記憶體位元給予一個唯一的編號 — 稱為記憶體位址(Memory Address)。如此一來，我們才能對電腦系統下達要對記憶體裡的哪一個位址進行存或取的操作。

1.2.2 次儲存體裝置

前一小節已說明過本書所指的主記憶體是指RAM而非ROM。由於無法長時間保存資料(不論是電腦系統關閉電源或重新啟動後資料都將不復存在)，再加上空間有限，無法儲存所有的資料。因此，我們需要使用一個非揮發性(Nonvolatile)的方式來長期地(不受電源關閉的影響)、大量地(相較於主記憶體大上許多倍的空間)保存資料 — 這就是我們需要次儲存體裝置(Secondary Storage Device)的主要理由[10)]。

常見的次儲存體裝置(Secondary Storage Device)包含軟式磁碟片[11)]、硬式磁碟機(Hard Disk Drive，HDD，又常簡稱為硬碟)、光碟片、記憶卡與隨身碟等。其中硬式磁碟機是最為常見的次儲存體裝置，普遍安裝在每一台個人電腦內，其容量高達數百GB到幾TB不等，可用以儲存大量的程式與資料，待需要時再載入到主記憶體內便可加以執行。光碟機也是非常普及的次儲存體裝置，程式與資料可以儲存在CD光碟片或DVD光碟片中，視需要再放入光碟機中進行讀取。近年來還有兩種新興的次儲存體：記憶卡與隨身碟。隨著數位相機、行動電話與PDA的普及，記憶卡已被廣泛採用為行動裝置的儲存體，不過在電腦系統必須安裝記憶卡的讀卡機才能讀取與寫入記憶卡。隨身碟在技術上與記憶卡類似，通常採用USB介面，無須再安裝其它裝置就可以直接進行操作。

通常在次儲存體裝置裡，程式與資料是以檔案(File)與目錄(Directory，又稱為資料夾)的形式存在及管理。一個檔案可以是一個可執行的程式或是可由程式加以處理的文字、影像、音樂等各式資料，同時必須儲存在特定的目錄中。一個目錄內可保存多個檔案，並且採用階層式的方式，在一個目錄中還可以有多個子目錄(Subdirectory)，每個子目錄中可保存多個檔案或子目錄。舉例來說，我們可以建立一個名為「Homework」的目錄來儲存所有的作業，並針對不同科目的作業再建立名為「English」、「Chinese」、「Math」或是「Programming」等的子目錄，各自保存相關的作業。換句話說，檔案是程式與資料的儲存的單位，目錄與子目錄則是管理檔案的方法。

1.2.3 中央處理單元

中央處理單元(Central Processing Unit，CPU)是儲存程式型電腦架構裡的控制單元(Control Unit)與算術邏輯單元(Arithmetic Logic Unit)的結合，負責協調整個電腦系統內的操作與資料的算術與邏輯運算。如前述，儲存在硬碟機中的程式必須先載入到主記憶體裡，再經由CPU來執行程式中的指令。CPU在程式指令的執行上，通常包含指令的讀取(Fetch)、解碼(Decode)、執行(Execute)與資料寫回(Write Back)等四大動作。更詳細來說，

- 讀取(Fetch)：CPU將儲存在主記憶體內的程式指令依順序加以讀取到其內部的指令暫存器(Instruction Register，IR)裡[指令暫存器(Instruction Register)是位於CPU內的高速記憶體空間，專門用以儲存程式指令。)]。

- 解碼(Decode)：接著對指令進行解碼，也就是將儲存在指令暫存器裡的程式指令的意義加以判讀。
- 執行(Execute)：依據前一階段的判讀結果，將程式指令加以執行，並將執行結果暫時保存於CPU內部的暫存器裡。
- 資料寫回(Write Back)：若需要保存執行結果，則將暫存器的內容寫回到主記憶體裡。

1.3 電腦軟體

電腦系統不能僅有硬體(Hardware)設備，還必須搭配適當的軟體(Software)才能提供我們各種功能與服務。舉例來說，單有一台 PlayStation電視遊樂器並不能提供你任何娛樂，你還必須放入遊戲光碟片[12)]才可以進行遊戲。此例當中的PlayStation遊戲主機與遊戲就分別是硬體與軟體，至於用以存放遊戲內容的光碟片只是所謂的載體(Carrier)。

所謂的電腦軟體(Computer Software)也是類似的概念，可以搭配電腦硬體來執行不同的工作。通常電腦軟體可以儲存在不同的在載體之上，例如前述例子中的光碟片，或是硬碟、隨身碟，又或者是直接儲存在雲端儲存裝置上[13)]。在資訊時代的今天，電腦這個名詞所涵蓋的範圍已經愈來愈廣泛，除了個人電腦(Personal Computer，PC)之外，包括智慧型手機(Smart Phone)、平板電腦(Tablet)、或是如智慧型手錶(Smart Watch)等新穎的穿戴式裝置(Wearable Device)、甚至是嵌入式系統(Embedded System)等，都可視為是「電腦」一詞的延伸。因此，現代的電腦軟體有時不單指可以在電腦上執行的軟體，有時也指那些可以在電腦之外的平台上執行的軟體。不過本書做為程式設計的入門書，後續的討論將僅侷限於個人電腦之上。

簡單的電腦軟體可能僅由一個程式負責執行所有的工作，較複雜的軟體則會由多個負責不同工作項目的程式來組成。電腦軟體還可以區分為系統軟體(System software)與應用軟體(Application Software)兩大類：

- 系統軟體：是為控制與指揮電腦系統運作而設計的軟體，例如作業系統、程式語言編譯器、驅動程式等。
- 應用軟體：根據使用者不同的應用需求所設計的軟體，例如文書處理軟體、試算表軟體、通訊軟體等。

通常系統軟體的開發牽涉較廣，尤其需要硬體方面的相關知勢才行，並不是初學程式設計者適合學習的；反觀應用軟體只與使用者的需求相關，也是初學者首先接觸與學習的標的。通常一個軟體可以由一個或多個程式(Program)組成，因此組成應用軟體的程式又被稱為應用程式(Application Program)。本書所使用的程式範例大多也都屬於此種類，可以在電腦系統上執行以幫助我們完成特定的工作。一個程式可以被視為是依據事先定義好的一組文字集合與語法規則(也就是依據程式語言的規定)所撰寫的程式碼的集合，在執行時先載入到電腦系統的主記憶體內，再由CPU加以執行[14)]。

1.3.1 系統軟體

所謂的系統軟體(System Software)主要負責控制、協調與管理在電腦系統內的硬體裝置，使其可以共同運作。常見的系統軟體包括作業系統(Operating System)、驅動程式(Device Driver)、組合語言(Assembly Language)[15)]的組譯器(Assembler)[16)]、各種程式語言(Programming Language)的編譯器(Compiler)[17)]、載入器(Loader)[18)]等。其中作業系統是負責管理電腦系統的硬體與軟體資源，包含主記憶體空間的管理與配置、工作的排程、輸入/輸出裝置的控制、磁碟機與檔案系統的管理等基礎核心的工作。通常一個作業系統還會伴隨著基本的電腦週邊裝置所需的驅動程式(Device Driver)以及載入器等必要的系統軟體，以提供使用者基本的電腦操作。常見的作業系統包含Microsoft Windows、Linux、Mac OS等，除了上述的核心工作外，這些作業系統也提供圖形化的使用者操作介面，以幫助使用者能以更便捷的方式與電腦系統互動。對使用者而言，像作業系統此類的系統軟體可以讓我們不用瞭解硬體運作的細節，就可以輕鬆地使用電腦系統。

1.3.2 應用軟體

所謂的應用軟體(Application Software)是依據特定的應用目的所開發的軟體，例如文書處理軟體、繪圖軟體、遊戲軟體等，使用者可以透過這些應用軟體來完成特定的工作。以Microsoft Windows視窗環境上常見的應用軟體為例Microsoft Word是一套文書編輯軟體，可以讓使用者進行文件的編輯、排版、列印等工作Microsoft Excel試算表軟體，可以讓使用者進行各式數值處理，以完成統計、報表製作等工作Microsoft PowerPoint則可以讓使用者用以製作簡報內容並用以進行簡報。此外，如果您要對數位相片進行處理，像Adobe Photoshop或Photo Impact等繪圖軟體就可以幫助您完成各式各樣的影像處理效果。如果您要進行的是大量資料的管理，也有MySQLMicrosoft SQL Server等資料庫應用軟體讓我們使用。甚至您想要用電腦系統來創作音樂作品，也有如Apple公司的GarageBand或是StudioOne等軟體可供你使用。當然更不用提，在電腦系統上還有許多遊戲軟體可供我們娛樂之用。

前述的軟體產品都是由廠商開發並公開販售供消費者使用，此外還有一些軟體是採取使用者訂製的方式，由軟體廠商或自行針對特定需求所加以設計，專供特定對象使用。例如連鎖的便利超商，就會自行或委外開發其專屬的電腦收銀系統、物流業者的倉儲與配送系統、製造業的物料與生產管理系統以及銀行業的帳戶系統等。這些客製化的系統通常所費不所費不貲，除了專屬的應用軟體之外，有時還必須包含特定的硬體裝置才能運作，例如便利超商櫃台的條碼機或是悠遊卡或一卡通的讀卡機。當然也有一些業者其營運規模尚未負擔得起客製化的專屬系統，因此商用現貨軟體(Commercial Off the Shelf(COTS)也就應運而生。軟體業者可以開發一般性的通用軟體，例如進銷存系統[19)]、會計系統、人事系統等，供中小型企業使用。

1.4 電腦程式

在前面的討論中，程式(Program)一詞已經多次地出現，可視為是軟體的組成單位。依據儲存程式型電腦架構的定義，一個程式的執行是由控制單元(Control Unit)每次從主記憶體中讀取程式當中的一個指令，然後再交由算術邏輯單元(Arithmetic Logic Unit)加以執行。這些被執行的指令，包含透過輸入裝置取得資料、要求算術邏輯單元進行各式運算、或者是透過輸出裝置將結果傳送出去。因此，我們也可以將程式定義為「一些指令的集合」，這些指令通常是特定的資料運算與邏輯處理，可交付給電腦系統來執行，以解決或滿足使用者的問題或需求，又因其必須在電腦系統上執行，因此又常被稱為電腦程式(Computer Program)在程式中所要執行的指令，對最早期的電腦系統來說，必須透過操作許多開關的開啟或關閉、或是改變機器的接線才能完成。現今的程式則是利用電腦程式語言(Programming Language)來編寫這些指令的集合，並且在符合儲存程式型電腦架構的電腦系統上，進行資料的輸入、處理與輸出，以解決或滿足使用者特定的問題或需求。例如一個程式可以用來幫助使用者換算華氏與攝氏的溫度、計算從台北到屏東的最短路徑，或者是讓使用者能夠進行文書處理、觀看影片或是聆聽音樂等動作。當然我們已經說明過，一個軟體是由一個或多個程式所組成，因此有時候一個簡單的程式就可以被視為是一個軟體，例如計算華氏與攝氏溫度的轉換就可能是一個僅由一個程式所組成的軟體。

在進一步說明之前，先讓我們認識以下兩個相關的名詞：程式設計(Programming)與程式語言(Programming Language)

1.4.1 程式設計/編寫程式碼

程式設計(Programming)就是指開發程式的過程，其中包含了程式的分析(Analysis)設計(Design)撰寫(coding)編譯(Compilation)與測試(Testing)等步驟。通常在職場上，負責開發程式的人就被稱做為程式設計師(Programmer)[20)]

至於程式語言(Programming Language)則是一組事先定義好的文字指令集合與語法規則，用以讓程式設計師據以編寫程式。因此，程式可視為遵循程式語言的語法規則所撰寫的程式碼(Code)[21)]的集合，可以在電腦系統上執行以解決或滿足使用者的特定問題或需求。程式設計師的主要工作，就是針對使用者的特定問題或需求，思索其解決方法，並遵循程式語言的語法規則來撰寫程式碼。由於撰寫程式這件事情，就是產生程式碼(Code)的過程，因此有時程式設計師的工作又被稱為coding寫程式或編程。

1.4.2 程式語言

如同人類使用自然語言(如國語、英語、日語等)來彼此溝通，程式設計師可以使用一種特別的電腦語言來和電腦溝通 — 程式語言(Programming Language)。只不過程式設計師和電腦的溝通是透過比較間接的方式，先依據程式語言的語法與語意規則將想要請電腦執行的工作寫成程式，然後再交由電腦系統執行，最後電腦系統透過輸出裝置讓我們得到執行的結果。以下本小節將就程式語言的種類分別加以說明：

1.4.2.1 第一代程式語言 — 機器語言

最早期的程式語言被稱為是第一代程式語言或是機器語言(Machine Language)。是在CPU上執行的指令集，又被稱為機器碼(Machine Code)。這些指令集與處理器緊密相關，不同的處理器需要搭配不同的指令集使用。程式設計師必須先熟記處理器指令的代碼及其意涵，才能夠編寫程式，此外還必須自行處理處理器內的暫存器(Register)以及記憶體內的資料，並且要記住處理器內各個單元的狀態；因此其程式的編寫相對困難度較高，尤其機器語言所編寫的程式碼是由0與1所組成的，對於人類來說實在沒有可讀性，維護上也十分不便。目前除處理器廠商外，已經非常少見到使用的機會了。

接下來，我們以32位元的MIPS處理器為例[22)23)]，為你說明並示範如何使用機器語言來完成簡單的加法與記憶體寫入。這個32位元的MIPS處理器，其機器語言規定每個要執行的機器碼都是32位元，且可分為R-type、I-type與J-type三種類型。以R-type為例，請參考figure 3。

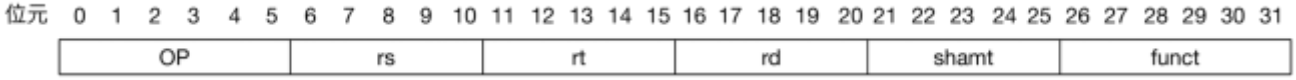

Fig. 3: 32位元MIPS處理器的R-type指令格式

在R-type的指令格式，最前面的6個位元(從0到5)與最後的6個位元(從26到31)是用以定義指令的操作(Operation)類型與功能(Function)。從位元6到20則分別以5個位元來表示該指令會使用到的三個暫存器，分別為rs、rt與rd。至於shamt則是shift amount的縮寫，表示需要位移的量(在本例中並沒有使用到)。假設我們要將register 1與register 2的值相加並將結果存放於register 3中，那麼可以使用下列的機器碼完成：

```
000000 00001 00010 00011 00000 100000
```

其中最前面與最後面的6個位元(也就是「000000」與「100000」)代表要進行加法，至於rs、rt與rd的值分別為「00001」、「00010」與「00011」，也就是十進位中的1、2與3，代表這個加法的運算是要將register 1與register 2的值相加並存放於register 3當中。

接下來，讓我們看看32位元MIPS處理器的I -type的指令格式，如figure 4所示，其中最前面的6個位元(從0到5)是用以定義指令的操作，後續從位元6到15則分別以5個位元來表示該指令會使用到的兩個暫存器，分別為rs與rt。至於address/immediate則是視操作的不同，所會使用到的記憶體位置或數值。

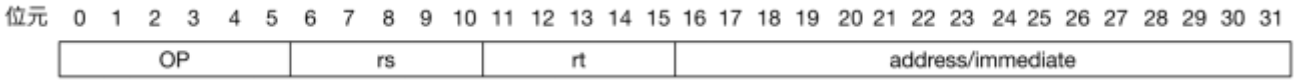

Fig. 4: 32位元MIPS處理器的I-type指令格式

現在。假設我們要將register 3的數值(也就是剛剛register 1與register 2相加的結果)，儲存到某一個記憶體位置中，那麼可以使用下列的機器碼完成：

```
100011 00100 00010 00000 00011 000100
```

其中最前面與最後面的6個位元(也就是「100011」)代表要將rs (也就是「00100」「register 3)內的值寫入到記憶體中，其位址由後續的rt內的值以及記憶體位址相加後決定，此例中的rt與記憶體位址分別為「00010」與「00000 00011 000100」，也就是register 4與196。假設register 4內的數值為4，則這個32位元的指令就是要將register 3內的數值寫入到記憶體位址200之處[24)]。這個例子雖然只是簡單的加法與記憶體存取，但因為使用機器語言所寫的程式是給處理器執行的機器碼，因此它是一系列0與1的組合，但實在令人難以親近。此外，機器碼不具備可移植性(Portability「亦可稱為可攜性)[25)]，無法將特定處理器上的機器碼移至其它不同的處理器執行[26)]。

1.4.2.2 第二代程式語言 — 組合語言

接下來第二代程式語言則是組合語言(Assembly Language)。它與機器語言同樣是與處理器緊密相關，但不直接使用處理器的指令集，而是使用一組助憶字(Mnemonic)來代表處理器指令，並可以使用標籤(Label)和符號(Symbol)來代表記憶體位址或數值，比起機器語言而言，已經多了許多彈性與可讀性。下面我們使用32位元MIPS處理器的組合語言，同樣將register 1與register 2的值相加，並將結果寫入到特定的記憶體位址中(同樣是196加上register 4之值的記憶體位址)，請參考下面的程式碼：

```
add $t3, $t1, $t2;
sw  $t3, 196($t4)
```

與前述的一堆0與1相比，現在這個例子看起來友善多了！其中第一行是將register 1與register 2的值相加後存放到register 3。與機器碼相比，只要使用add指令就可以完成，不再需要去記憶或是查詢其機器碼中的操作與功能該如何撰寫，而是使用有意義的add來代替；其後的$t3、$t1與$t2則是MIPS的組合語言規定用以表示register 3、1與2的方式，其語法規定以「$t」開頭，後面接上欲指定的register編號即可。因此，我們可以把第一行程式碼視為是進行「register 3 = register 1 + register 2」的運算，這樣是不是容易多了？至於第二行的「sw」則是store word的縮寫[27)]，表示要將特定register的值寫入到特定的記憶體位置中，後面接著的「$t3, 196($t4)」即是要將register 3的值寫入到記憶體位址196加上register 4的值，也就是200之處。

要注意的是這個組合語言的程式碼範例並不能直接交由處理器去執行，畢竟處理器唯一能執行的只有機器碼而已，因此我們還需要一個工具來將組合語言的程式碼轉換為可在處理器上執行的機器碼 — 這個工具被稱為組譯器(Assembler)[28)]。

相對於機器碼一詞，我們在習慣上會將原始的、還未進行轉換的組合語言的程式碼，稱為原始程式(Source Code)。雖然組合語言比起機器語言已經簡單許多，但是程式設計師還是必須瞭解並熟悉處理器的內部架構與記憶體等相關細節，例如處理器內的暫存器、旗標以及記憶體定址等，而且由於處理器的差異，不同處理器的組合語言的差異也很大，一個針對特定處理器所撰寫的程式碼，將不能套用在其它的處理器之上，換言之組合語言和機器語言一樣缺乏可移植性；有時甚至會發現在某個處理器上的組合語言寫程式的經驗，可能完全不能應用到另一個處理器之上。

此外，從本質而言，機器語言與組合語言幾乎是相同的，都是與機器緊密相關的語言，差別只在於組合語言提供了一種較容易的方法來產生機器碼而已。與人類所使用的自然語言相比，機器語言與組合語言所能表達的僅限於與處理器相關的操作；換句話說，就好像是透過機器語言與組合語言的程式來操作處理器與記憶體。由於這兩種語言都是對處理器等硬體裝置進行操作，因此機器語言與組合語言又被合稱為低階程式語言(Low-Level Programming Language)[29)]。雖然這兩種語言被稱為低階程式語言，但其執行效能反而比較好。這是因為低階程式語言與處理器的指令直接相關，其所產生的機器碼可以直接在處理器上執行，其效能為其它程式語言所不能比擬的。

1.4.2.3 第三代程式語言 — 高階程式語言

前面介紹的兩個低階程式語言(包含機器語言與組合語言)都是與處理器緊密相關的語言，也就是必須具備處理器相關的知識才能進行程式的撰寫，同時在不同處理器間也不具可移植性。相對的，第三代程式語言與處理器無關，我們不需要瞭解有關處理器等硬體的知識，就可以進行程式的撰寫，其所產生的程式也與處理器無關。相較於機器語言與組合語言，使用第三代程式語言其所撰寫出的程式碼不但較容易理解，而且其可移植性也較高，通常僅需要進行小幅度的修改甚或是完全不需要修改，就可以在不同的電腦系統上執行。第三代程式語言不但與處理器無關，並且使用人類較容易學習與使用的文字與符號進行程式的撰寫，因此也被稱之為高階程式語言(High-Level Programming Language)。

使用高階程式語言所撰寫的程式，被稱做為原始程式(Source Code)。並不能直接交付電腦系統執行，必須先經過轉換的過程才能產生可以在電腦系統上執行的可執行檔。這種轉換的過程有兩種方法：編譯式(Compilation)與直譯式(Interpretation)。編譯式並不是直接將程式轉換成可執行檔，而是先轉換產生所謂的目的碼(ObjectCode)。然後再由聯結程式(Linker)將所產生的目的碼與相關的函式庫(Library)進行聯結，如此才能夠產生可執行檔。至於直譯式則是採用互動的方式，每一行原始碼都會立刻進行轉換並加以執行，而不會產生目的檔或可執行檔。其優點在於程式設計師可以立即看到每一行程式碼的結果與做用，但缺點是每一次執行都必須重新進行轉換的動作，其執行效率較差。我們把負責編譯與直譯動作的程式，分別稱為編譯器(Compiler)與直譯器(Interpreter)。

第一個高階程式語言是在1940年代初期，由德國電腦科學家Konrad Zuse所設計的Plankalkül語言，然而受到第二次世界大戰的影響。Plankalkül語言並未完成其實作，也未向世人公開；直到1948年。Konrad才將Plankalkül語言發表於Archiv der Mathematik期刊[30)]，並遲至1975才由Joachim Hohmann完成Plankalkül語言的實作[31)]。由於在1940年代初期，主流的程式語言仍是機器語言與組合語言這類的低階程式語言，人們很難接受Plankalkül語言的全新概念，再加上當時並沒有實作版本，因此Plankalkül語言並沒有受到應有的重視。

直到1950年代晚期，高階程式語言的發展才獲得重視。1957年，美國電腦科學家John Backus為IBM公司的IBM 704系統所設計的Fortran語言(FORmula TRANslation)[32)]，才成為了第一個廣為人知的高階程式語言，並且在科學與工程計算領域中被廣泛地使用。1958年，美國電腦科學家John McCarthy教授在Massachusetts Institute of Technology (MIT。麻省理工學院)時發表了Lisp語言[33)]，不但被廣泛應用於人工智慧領域，同時也是公認的第二個被普遍接受的高階程式語言。其後在1960年，美國電腦科學家Grace Murray Hopper[34)]針對商業應用領域，發表了一個名為COBOL(Common Business Oriented Language。通用商業導向語言)的程式語言。不論是Fortran。Lisp或COBOL都為後續其它程式語言的發展奠定了良好的基礎，實至今日也仍有電腦系統還在使用這些程式語言所撰寫的程式。

接著在1960年代初期至1970年代初期，開始有許多高階程式語言陸續被發表，較著名的包含了ALGOL[35)]。Simula[36)]。CPL[37)]。BCPL[38)]。BASIC[39)]。PL/1[40)]等程式語言，其中的CPL與BCPL就是後來發展出的C語言的前身。高階程式語言的發展一直以來都未曾停歇，在1970年代有Pascal。C。Smalltalk與Prolog等、1980年代則有Ada。C++。Perl等、1990年代則有Python。Visual Basic。Ruby。Java。Delphi。JavaScript及PHP等，以及從2000年開始迄今，還有如C#。Go與Swift等程式語言不斷推陳出新。這些高階程式語言依其特性，可概分為兩類：程序導向(Producedure-Oriented)以及物件導向(Object-Oriented)。

- 程序導向程式語言(Producedure-Oriented Programming Language):以程序為主軸，將所欲完成的功能或欲解決的問題，編寫成一段一段的程式碼，並依據安排好的步驟逐一加以執行，其中也可以包含條件式的程序[41)]以及迴圈式的程序[42)]。此類以程序為導向的程式語言包含C語言以及FORTRAN。COBOL。BASIC。Pascal。Perl。PHP等皆屬之，是當前資訊產業從業人員所必備的專業技能之一。
- 物件導向程式語言(Object-Oriented Programming Language)。以物件為主軸，針對欲完成的功能或欲解決的問題所存在的環境，包含人、事、時、地、物等，都以抽象的物件來進行對應，其中每個物件係由相關資料以及操作資料的方法所組成。透過程式中的物件與物件，或是物件與使用者之間的互動，物件導向程式就可以來達成特定的程式功能。物件導向程式語言，不同於程序導向的程式語言，它還支援資訊封裝、繼承以及多型等概念，是目前主流的開發語言。常見的物件導向程式語

言包括C++、C#、Java、Python以及Ruby等，也是當前資訊產業人員必須具備的專業技能之一。

截至目前為止，高階程式語言的數量已經達到數千種以上，並且仍然在不斷地增加當中。如果貪心地想要將世上全部的程式語言都學會，這實在是一件不可能的任務！以大學的資訊工程學系為例，在四年的學習過程中，課堂上講授的也只有少數幾個比較重要的程式語言，例如C、C++、C#、Java以及PHP等。在這些程式語言中，C語言可算是最為基礎也最為重要的一個程式語言，因為C語言不但是當前最普遍被使用的程式語言之一，同時C語言的語法、語意與相關的程式設計概念，都可以延伸到很多其它的程式語言中；包含在C++、C#、Java以及PHP等程式語言當中，都可以看到很多和C語言相同之處。而本書的主角C++語言，除了具有承襲自C語言的程序導向特性之外，還額外具有物件導向的特性，是目前最為主流的程式語言之一。因此，對於有志從事資訊產業的學生而言，C++語言的學習至關重要。

以下我們以C++語言為例，示範如何進行一個矩形面積的計算：

```
int area, width, height;
width = 10;
height = 20;
area = width * height;
```

在上面這段程式碼當中，第一行是說明程式會使用到三個整數area、width與height，分別代表矩形的面積、寬與高；第二行與第三行則是設定該矩形的寬與高的值；最後一行則是計算這個矩形的面積。當然，上面的程式碼還少了許多細節，但與本章前面的組合語言程式範例相比，你會發現這段高階程式語言的程式碼似乎很容易理解，因為它與我們人類的思維一致。雖然你還沒開始學習C++語言，但相信你看完了這個範例的程式碼後，應該能夠略懂一二。

1.4.2.4 第四代程式語言 — 極高階程式語言

第四代程式語言最早源起於1970年代，其發展目標是要提供比起第三代程式語言更為高階、更接近人類思維或自然語言的程式語言，因此又被稱為極高階程式語言(Very High-Level Programming Language)。為了實現這個目的，第四代程式語言通常針對特定應用領欲開發，包含資料庫管理、報表產生、使用者圖形界面設計或是網頁開發等，例如應用在資料庫系統的查詢語言Structured Query Language(SQL)就是一例。下面的例子就是使用SQL語言，對一個名為Students的資料表格下達列示所有及格學生名單的要求：

```
SELECT Name FROM Students WHERE score>=60;
```

如何？是不是很接近英語呢？由於第四代語言接近人類所使用的自然語言，因此具有容易學習與易於使用的優點。

1.4.2.5 第五代程式語言 — 自然語言

第五代程式語言又稱為自然語言(Natural Language)。其發展目標就是讓我們可以直接使用自然語言，對電腦系統下達命列以完成工作。第五代程式語言的發展與人工智慧領域緊密相關，它需要在接收到使用者的命令後，透過具有智慧的軟體程式去解讀使用者命令的意涵，才能讓電腦系統提供正確的服務。一些成功的案例是以程式語言的函式庫或其它方式存在，可以讓我們在高階程式語言的程式中把使用者所輸入的自然語言(包含文字或語音)進行解讀與後續的處理，例如GATE(general architecture for text engineering)[43] 或是MARF(modular audio recognition framework)[44]。除了在程式設計方面的應用，也有一些軟體服務是以直接提供使用者操控電腦系統為目的，例如Apple iOS device上的Siri。或是在Android phone上面的OK Google服務。

1.4.3 軟體/程式開發工具

依所使用的程式語言的不同，其開發流程可以概分為直譯式(Interpretation)與編譯式(Compilation)兩類，其中直譯式的程式語言通常是在專屬的環境中進行程式的開發，並且所寫的每一行程式都會在該環境中解譯並加以執行。這個用以解譯與執行直譯式程式語言的軟體被稱為直譯器(Interpreter)。至於在編譯式方面，我們需要一個文字編輯器(Text Editor)軟體來撰寫及修改原始程式(Source Code)。然後使用一個被稱為編譯器(Compiler)的軟體，來將原始程式轉換為目的碼(Object Code)。再經由聯結程式(Linker)將程式執行所需要的函式等加以結合後，產生出可以在電腦系統上執行的可執行檔(Executable File)。其中將原始程式轉換為目的碼的過程，又被稱為編譯(Compile)。現在讓我們回顧編譯式程式語言的開發過程，請參考figure 5。

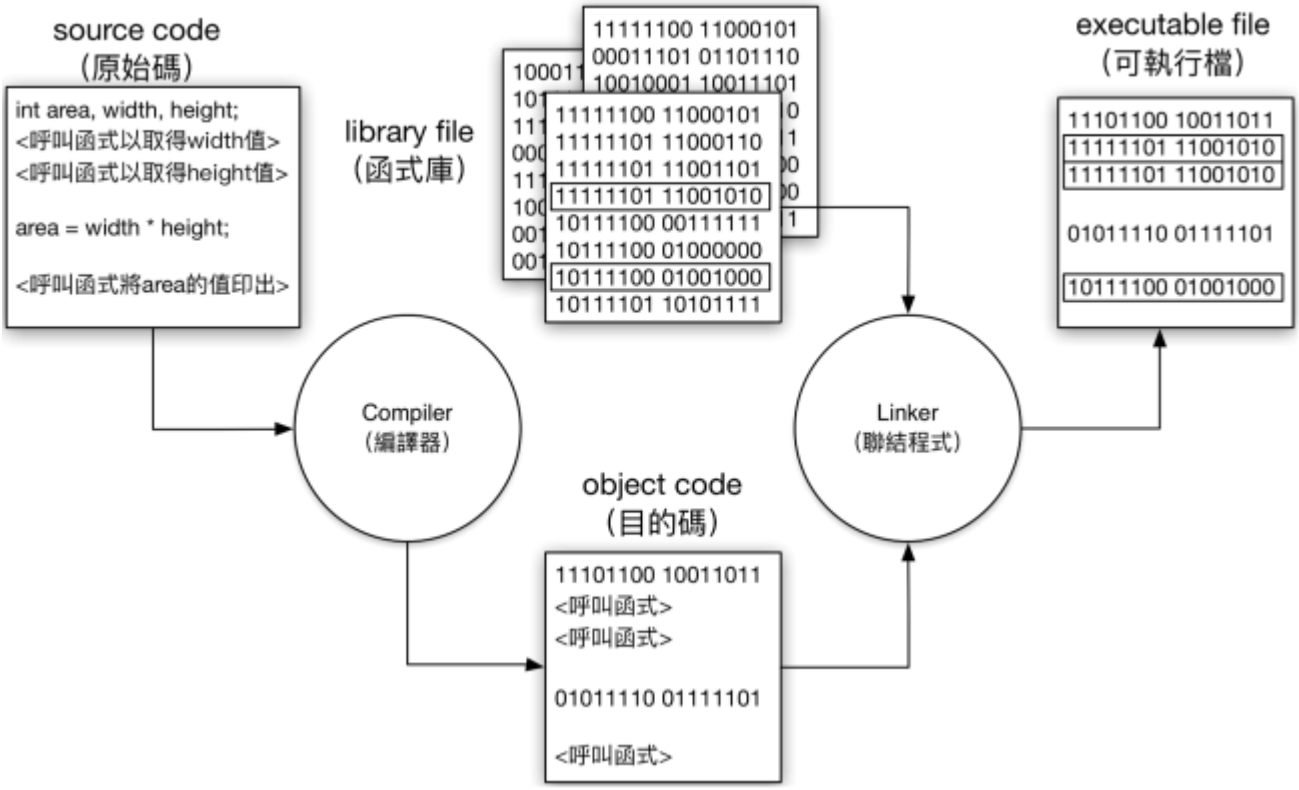

Fig. 5: 典型高階程式語言的程式開發流程

如figure 5，我們將開發高階程式語言的程式時，會使用到的軟體工具說明如下：

- 用以編輯原始程式的文字編輯器(Text Editor)。只要是能夠編輯文字的軟體皆可，就算是Windows系統中的「記事本」也可以，不過程式設計師通常會選擇一些專門設計用來寫程式的文字編輯軟體，上面會提供行號顯示、自動縮排、語法高亮(突顯)等對寫程式有幫助的功能；有的還會提供專案/目錄管理、甚至是編譯與執行等功能。常見的選擇包含了跨平台(Cross-Platform Software)[45]的Atom[46]、Visual Studio Code[47]與Sublime Text[48]，以及Microsoft Windows專屬的Notepad++[49]。除了這些可用於程式設計的文字編輯器軟體之外，也有許多程式設計師直接使用在伺服器(Server)[50]上的vi、vim、emacs、joe、pico等軟體；如果你還沒有使用伺服器的經驗，那麼這次學習C++語言將會是一個難得的機會，可以要求自己在伺服器上進行C++語言的程式設計，在學習程式語言的同時，也可以順便熟悉伺服器的操作環境。
- 用以產生目的碼的編譯器(Compiler)。大部份的高階程式語言都是屬於編譯式的程式語言，都需要有編譯器才能將原始程式碼轉換成目的碼。由於編譯器的好壞對於所產生的目的碼(乃至於最終產生的機器碼)而言十分重要，我們必須謹慎地選擇所需的編譯器。通常一個程式語言會有多個不同的編譯器可供選擇，程式設計師必須依其價格、功能、效率、作業系統等因素進行選擇。
- 用以聯結程式語函式庫來產生可執行檔的聯結程式(Linker)。此軟體的作用是將目的碼與其所使用到

的函式庫整併並產生可執行檔。目前，大部份的高階程式語言的編譯器都已內建了聯結程式的功能，或是在編譯完成後自動接續執行聯結程式的功能。

在典型的開發流程中，我們除了以文字編輯器進行原始程式的撰寫外，還要透過編譯器與聯結程式才能產生可執行檔。專業的程式設計師還會搭配整合開發環境(Integrated Development Environment，IDE)進行程式的開發；所謂的IDE可以讓程式設計師在同一套軟體中，進行原始程式的編輯以及編譯的軟體，除此之外IDE還提供檔案或專案管理、除錯以及建置等工具，這些開發程式所需要的軟體工具通通都整合在同一個軟體當中，對於程式設計師來說，IDE是個十分便利的開發環境。常用的IDE包含有NetBeans、Eclipse、Dev-C++、Xcode以及Microsoft Visual Studio等。

在職場上，為提升開發軟體的效率，各家公司與軟體團隊往往都會使用IDE來開發軟體；不過對於初學者來說，筆者建議大家還是先不要在一開始就使用IDE來進行程式的設計，免得對於最基礎的程式開發流程以及細節都沒有足夠的認知，對於未來更進階的學習造成負面的影響。「一分耕耘，一分收獲」這是最簡單、也最通用的道理，請先以最基礎的方式進行程式設計的學習，並從當中為自己累積足夠的經驗後，再開始以IDE環境進行軟體的設計與開發，這樣才是循序漸進的學習良方！

<nowiki><nowiki><nowiki><nowiki>C++</nowiki></nowiki></nowiki></nowiki>

---

[1] 例如在Terminator系列電影(台譯「魔鬼終結者」)中的Skynet(天網)系統。
[2] 例如Matrix系列電影(台譯為「駭客任務」)中的Martix(母體)。
[3] 可參閱http://www.bbc.com/news/technology-30290540。
[4] 機器碼(Machine Code)通常係指可以在處理器硬體上直接執行的指令，又稱為機器指令。
[5] J. von Neumann, First Draft of a Report on the EDVAC, Technical Report, Moore School of Electrical Engineering, University of Pennsylvania, June 30, 1945.
[6] J. von Neumann, First Draft of a Report on the EDVAC, IEEE Annals of the History of Computing, Vol. 15, Issue 4, pp. 27-75, October 1993.
[7] M. D. Godgrey and D. F. Hendry, The Computer as von Neumann Planned It, IEEE Annals of the History of Computing, Vol. 15, Issue 1, pp. 11-21, January 1993.
[8] 一般認為在John von Neumann於1945年發表論文之前，儲存程式型電腦架構已在賓州大學的摩爾電機學院內流傳了，甚至早在1936年Konrad Zuse所提出的專利就已出現此一概念的雛型。
[9] 關於儲存程式型電腦架構的概念另有一說是John Presper Eckert於1943年所創(可參考H. Lukoff, From Dits to Bits: A Personal History of the Electronic Computer, Robotics Press, 1979.)。
[10] 雖然唯讀記憶體(Read-Only Memory，ROM)也是屬於非揮發性的儲存空間，但其無法寫入資料且空間通常較小，所以並不適用。
[11] 現在已經幾乎看不到軟式磁碟機的蹤影，但目前美國國防部的核武控制系統仍在使用此種舊式的軟式磁碟機，至今已超過五十年以上。
[12] 除了遊戲光碟片外，PlayStation也提供線上下載遊戲軟體的功能。
[13] 如Goolge雲端硬碟、Apple公司的iCloud、Dropbox等服務，皆利用網路提供內容存取的服務，使用者只要透過網路就可以存取檔案。
[14] 通常一個軟體還可能需要先從載體安裝到電腦系統上的次儲存體裝置後，才能加以執行。
[15] 組合語言係用於電腦系統或其它可程式化的裝置(如微處理器、微控制器等)上的一種低階程式語言，其程式碼通常與機器指令有一對一的對應關係。
[16] 組譯器(Assembler)可將使用組合語言所撰寫的程式，轉換成為可在處理器上執行的機器指令。
[17] 編譯器(Compiler)泛指可將使用一般高階程式語言所撰寫的程式，轉換為可在處理器上執行的可執行檔。
[18] 載入器(Loader)係負責將程式或軟體，自其載體(Carrier)載入到特定的記憶體位置，以供後續執行使用。
[19] 進銷存系統係指可以協助企業進行包含進貨、銷售與存貨等工作的應用軟體。
[20] 現代資訊產業分工很細，有時僅將負責編寫程式碼的人稱為程式設計師。至於負責分析與設計的人，則被稱為系統分析師；負責測試工作的人則稱為測試工程師。
[21] 習慣上，我們將依據程式語言所撰寫的程式視為是程式碼的集合，其中code意指一行一行的程式碼。
[22] MIPS Technologies Inc., MIPS32 4K Processor Core Family Software User’s Manual, Document Number: MD0016, Revision 1.17, September 25, 2002.

[23] Dominic Sweetman, See MPIS Run, 2nd Edition, Morgan Kaufmann Publishers, ISBN 978-0-12-088421-6, October 2006.
[24] 此處的196位址以及register 4的值，可分別稱基底位址(Base Address)與位移(Offset)。透過基底位址加上位移的值，即可得到目的記憶體位址。
[25] 可移植性係指同一程式在不同電腦系統上執行的可行性，可移植性高的程式語言表示其所撰寫出的程式可以容易(甚至完全不用修改)的在不同的電腦系統上執行。
[26] 當然，支援相同指令集的相容處理器不在此限。
[27] Store word表示要儲存一個word(字組)的內容到記憶體中，其中word為記憶體空間的單位，在MIPS處理器中，一個word為4個bytes。也就是32位元。
[28] 事實上，組譯器並不會產生可以執行的檔案，而是產生所謂的目的檔(Object File)。然後再由聯結程式(Linker)將目的檔與相關的函式庫(Library)聯結產生可執行檔(Executable File)。
[29] 所謂的低階(Low-Level)係一種相對的概念，指得是電腦系統與使用者(也就是人類)之距離；愈接近電腦系統底層的硬體裝置就愈低階；反之，愈接近使用者的思維與習慣就愈高階。
[30] Konrad Zuse, Über den Allgemeinen Plankalkül als Mittel zur Formulierung schematisch-kombinativer Aufgaben, Archiv der Mathematik, Volume 1, Issue 6, pp. 441-449, November 1948.
[31] Joachim Hohmann, Der Plankalkül im Vergleich mit algorithmischen Sprachen, Reihe Informatik und Operations Research, S. Toeche-Mittler Verlag, Darmstadt, 1979.
[32] Fortran語言的命名是取自FORmulation TRANslation的縮寫，中文一般直譯為「公式轉換」，或是音譯為「福傳」語言。
[33] John McCarthy, Recursive Functions of Symbolic Expressions and their Computation by Machine, Part I, Communications of the ACM, Volume 3, Issue 4, pp. 184-195, April1960.
[34] 著名的女程式設計師，除了設計出廣為人知的COBOL語言外，也是「debug(程式除錯)」一詞的創始者。
[35] ALGOL為ALGOrithmic Language的縮寫，顧名思義ALGOL是用以撰寫開發演算法為主的程式語言，其著名的版本為ALGOL 58、ALGOL 60與ALGOL 68。
[36] 1962年由Ole-Johan Dahl與Click-Sd於挪威電子計算機中心開發，承續了ALGOL 60的基礎，並被公認為第一個物件導向程式語言。
[37] CPL全名為Combined Programming Language，是在1960年代初期，由英國University of Cambridge(劍橋大學)與University of London(倫敦大學)合作開發的，因此CPL又被戲稱為Cambridge Plus London。CPL受到ALGOL的影響很深，但不同於ALGOL主要用於演算法的開發。CPL所追求的是更廣的應用領域，甚至支援使用低階程式語言，也因此CPL在當時是一個十分龐大的程式語言。
[38] BCPL全名為Basic Combined Programming Language，是由英國University of Cambridge於1966年所發表的程式語言，其開發受CPL影響，是第一個使用「{}」做為程式區塊的程式語言。其後續所推出的B語言即為C語言之前身，因此又有人將BCPL戲稱為Before C Programming Language。
[39] Basic是Beginner’s Beginner's All-purpose Symbolic Instruction Code的縮寫，是一套著名的直譯式程式語言，中文一般譯為「培基」語言。歷經多年的發展。Basic語言有許多版本流傳，台灣許多高職生的第一個程式語言就是微軟公司所推出的「Microsoft Visual Basic」語言。附帶一提。Basic語言原本應該寫做「BASIC」語言，但因為Microsoft Visual Basic過於流行之故，現在大家都寫做「Basic」了。
[40] PL/1全名為Programming Language One，是由IBM公司於1964年所發表，用於工程科學計算、商業運算及系統程式等領域。
[41] 條件式的程序會伴隨著一個條件，在執行時必須依據該條件的結果，決定該執行的程式碼為何。程式設計師可以使用條件式的程序，針對不同情況給定不同的程式碼加以執行。
[42] 迴圈式的程序則是可以指定特定的程式碼，依指定的方式反覆地執行，對於重複性的工作非常有幫助。
[43] 可參考http://gate.ac.uk。
[44] 可參考http://marf.sourceforge.net。
[45] 所謂的跨平台軟體(Cross-Platform Software)係指該軟體可在多個不同平台上執行，通常包含了Microsoft Windows、Mac OS與Linux等常用的作業平台。
[46] 可參考https://atom.io。
[47] 可參考https://www.visualstudio.com/products/code-vs。
[48] 可參考https://www.sublimetext.com。

[49] 可參考https://notepad-plus-plus.org

[50] 伺服器(Server)泛指硬體規格較一般個人電腦高階，可透過網路連接提供多用戶同時使用的電腦系統。

From:
https://junwu.nptu.edu.tw/dokuwiki/ - **Jun Wu的教學網頁**
國立屏東大學資訊工程學系
CSIE, NPTU
Total: 260066

Permanent link:
https://junwu.nptu.edu.tw/dokuwiki/doku.php?id=cppbook:ch-csandpl

Last update: **2024/01/12 07:27**