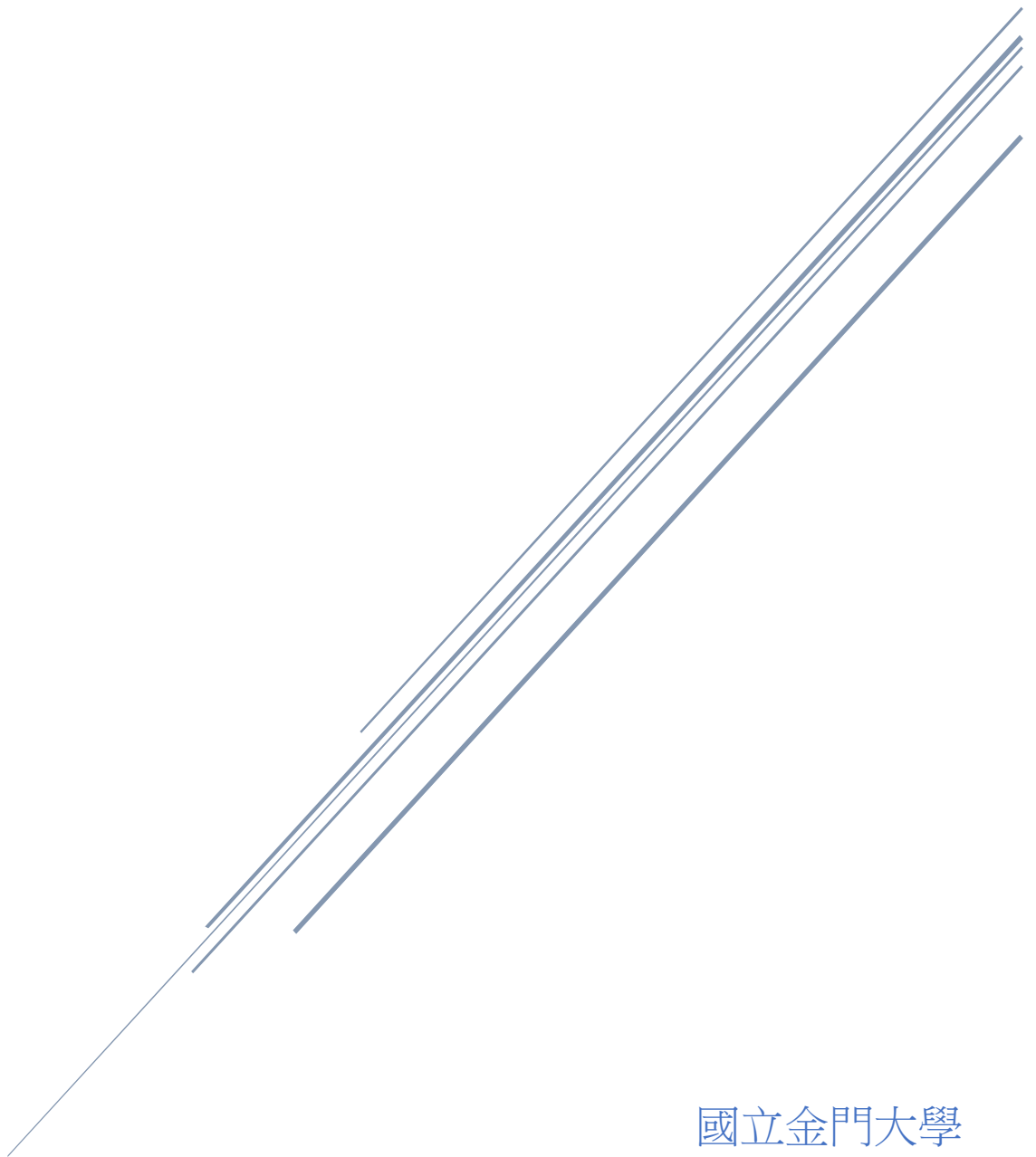


資訊工程 學號:110710544 姓名:陳詠鑫

系統程式-期中報告 主題: 編譯器研究 (C)



國立金門大學

系統程式課程

## 目錄

什麼是編譯器編譯器:	2
工作流程:	3
主要工作:	4
編譯器的分類:	5
一般編譯器可以分為以下兩類：	7
歷史:	11
在 Windows 上建立 C 語言開發環境	12
前言:	12
C 編譯器	13
Visual C++	13
Cygwin 內的 GCC	15
Windows Subsystem for Linux 內的 GCC	16
MinGW + MSYS	17
編輯器或 IDE	18
Visual Studio	18
CLions	19
Code::Blocks	19
KDevelop	19

Dev-C++ .....	20
其他編輯器.....	21
管理 C 專案 .....	22
安裝第三方函式庫 .....	23
如何選擇？ .....	24
使用 MinGW (MSYS2) 搭配 Code::Blocks.....	25
如果 Code::Blocks 無法抓到 GCC (MinGW)的路徑.....	26
選擇專案類型，這裡選「Console application」： .....	27
你所不知道的 C 語言：編譯器原理和案例分析 .....	30
編譯器和軟體工業強度息息相關.....	31

## 什麼是編譯器編譯器:

編譯器（compiler）是一種電腦程式，它會將某種程式語言寫成的原始碼（原始語言）轉換成另一種程式語言（目標語言）。

主要的目的是將便於人編寫、閱讀、維護的進階電腦語言所寫作的原始碼程式，翻譯為電腦能解讀、執行的低階機器語言的程式，也就是執行檔。

編譯器將原始程式（source program）作為輸入，翻譯產生使用目標語言（target language）的等價程式。

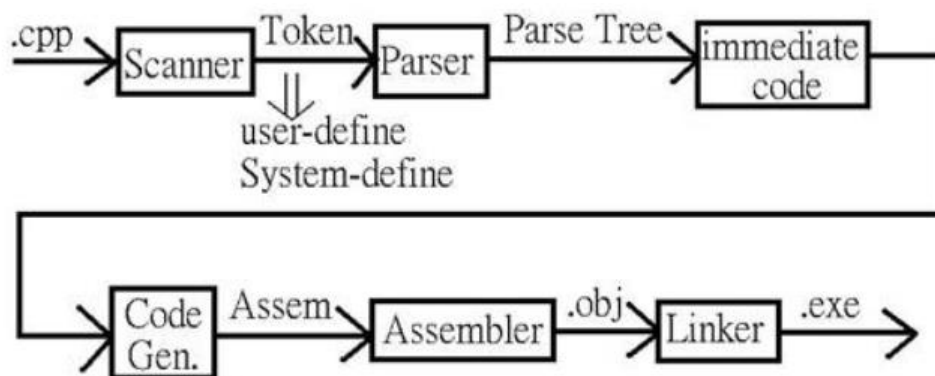
原始碼一般為高階語言（High-level language），如 Pascal、C、C++、C#、Java 等，而目標語言則是組合語言或目標機器的目的碼（Object code），有也稱作機器碼（Machine code）。

## 工作流程:

一個現代編譯器的主要工作流程如下：

原始碼 (source code) → 預處理器 (preprocessor) → 編譯器 (compiler)  
→ 組譯程式 (assembler) → 目的碼 (object code) → 連結器 (linker) →  
執行檔 (executables)，最後打包好的檔案就可以給電腦去判讀執行了。

編譯器的主要工作流程圖表



## 主要工作:

主要工作:「預先處理」(Preprocess)、「編譯」(Compile)和「組譯」(Assemble)。

預先處理即是做一些在編譯前要做的工作，之後就進行編譯。

在編譯過程中，編譯器會檢查程式有沒有錯誤，而錯誤主要有兩類：

「語法錯誤」(SyntaxError)和「語義錯誤」(SemanticError)。

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，那麼編譯器就不懂得「看」你的程式。而語義錯誤就好像串錯了字，例如你用了未經宣告的變數，就算編譯器懂得看，也不懂得編譯你的程式。

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，編譯器會把程式內每個句子轉成更低階的方式，一般來說是指「組合語言」(Assembly)方式。

轉成組合語言後，組譯過程就會把每個組合語言句子轉成「機械語言」(MachineCode)方式（通常編譯器都內置了組譯器），這種方式稱為「目的碼」(ObjectCode)，產生另一個檔案"file.obj"。

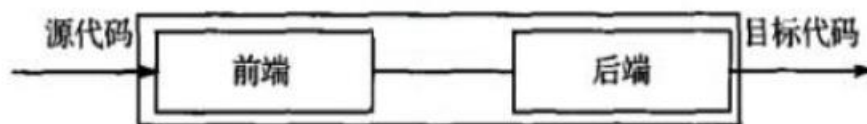
最後到了「連結」(Link)過程，就會把你的程式、有關的程式和程式庫所產生出來的\*.obj，轉成可以在電腦上執行的方式，產生另一個檔"file.exe"，這個檔案就可以執行了。

## 編譯器的分類:

典型的編譯器輸出是由包含入口點的**名字**和**地址**以及**外部調用**的機器代碼所組成的目標文件。

一組目標文件，不必是同一編譯器產生，但使用的編譯器必須採用同樣的輸出格式，可以鏈接在一起並生成可以由用戶直接執行的可執行程式。

在運行過程中，編譯器又可分成只依賴於**源語言**的**編譯器前端**和只依賴於**目標語言**的**編譯器後端**兩大部分，編譯器前後端結構如圖



編譯器前後端示意圖

前端主要負責解析(parse)輸入的源程式，由詞法分析器和語法分析器協同工作。詞法分析器負責把源程式中的“單詞”(Token)找出來，語法分析器把這些分散的單詞按預先定義好的語法組裝成有意義的表達式、語句、函數等。

例如“**a=b+c**”，前端詞法分析器看到的是“**a='b'+c**”，語法分析器按定義的語法，先把它們組裝成表達式“b+C”，再組裝成“a=b+c”的語句。前端還負責語義(semantic checking)的檢查，例如檢測參與運算的變數是否是同一

類型的，簡單的錯誤處理。最終的結果常常是一個抽象的語法樹 AST(Abstract Syntax Tree)，這樣後端可以在此基礎上進一步優化處理。編譯器後端主要負責分析、優化中間代碼以及生成機器代碼。一般來說所有的編譯器分析、優化、變型都可以分成兩大類：函數內進行和函數間進行。

很明顯，**函數間的分析**優化更準確，但需要更長的時間來完成。



## 一般編譯器可以分為以下兩類：

① “本地” 編譯器：編譯器可以生成用來在與編譯器本身所在的電腦和操作系統(平臺)相同的環境下運行的目標代碼。

② 交叉編譯器：編譯器也可以生成用來在其他平臺上運行的目標代碼，交叉編譯器在生成新的硬體平臺時非常有用。

交叉編譯這個概念的出現和流行是和嵌入式系統的廣泛發展同步的。

在進行嵌入式系統的開發時，運行程式的目標平臺通常只有有限的存儲空間和運算能力。

例如常見的 ARM 平臺，其一般的靜態存儲空間為 16~32MB，而處理器的主頻為 100~500MHz。

這種情況下，在 ARM 平臺上進行本機編譯就不太可能了，這是因為一般的編譯工具需要很大的存儲空間，並需要很強的處理器運算能力。為了解決這個問題，交叉編譯工具就應運而生了。通過交叉編譯工具，就可以在 CPU 能力很強、存儲控制項足夠的主機平臺上(例如通用電腦)編譯出針對其他運行平臺的可執行程式。

在一種電腦環境中運行的編譯程式，能編譯出在另外一種環境下運行的代碼，  
這個編譯過程就叫交叉編譯。

也就是在主機平臺上開發程式，併在這個平臺上運行交叉編譯器，編譯出程  
式，這個編譯程式將在目標平臺上運行。

這裡需要註意的是所謂平臺，實際上包含兩個概念：體繫結構和操作系統。

同一個體繫結構可以運行不同的操作系統；同樣，同一個操作系統也可以在不  
同的體繫結構上運行。舉例來說，常說的 X86Linux 平臺實際上是 IntelX86 體  
繫結構和 LinuxforX86 操作系統的統稱；而 X86WinNT 平臺實際上是 IntelX86  
體繫結構和 WindowsNTforX86 操作系統的簡稱。

在交叉編譯技術中有兩種比較典型的實現，一個稱之為 Java 模式，即 Java  
的位元組碼編譯技術；另一個稱之為 GNUGCC 模式，即通常所講的 CrossGCC  
技術。Java 模式(如圖所示)的最大特點是引入了一個自定義的虛擬機，即 Java  
虛擬機 JVM(Java Virtual Machine)。

所有 Java 源程式都會首先被編譯成只在這個虛擬機上才能執行的“目標代  
碼”：位元組碼(Bytecode)。

在實時運行時，可以有兩種運行方式：一種是編譯所獲得的位元組碼由 JVM 在  
實際電腦系統上執行；另一種方式是通過 Java 實時編譯器(Just-In-  
TimeCompiler)將位元組碼首先轉換成本地機可直接執行的目標代碼，而後交  
給實際的電腦系統運行。

這實際上是一個兩次編譯過程，一次是非實時的，一次是實時的。

由於第一次是非實時編譯，Java 編譯器生成的是基於 JVM 的“目標代碼”，

可以將它的編譯技術也稱為交叉編譯。

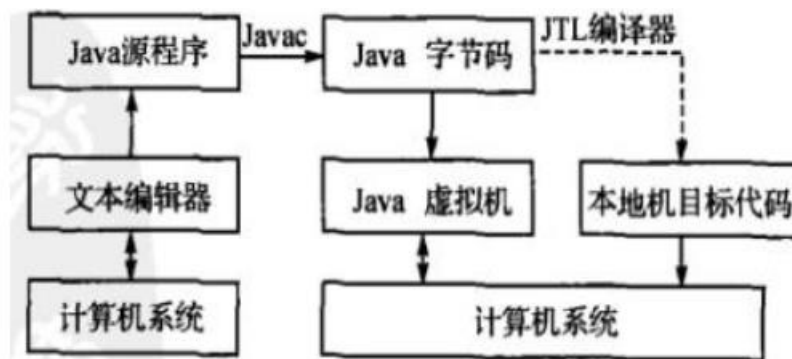


图 Java 模式交叉编译

GCC 模式(如圖所示)通過 CrossGCC 直接生成目標平臺的目標代碼，從而能夠

直接在目標平臺上運行。

這裡的關鍵是 CrossGCC 的生成和選擇問題。

需要根據目標平臺的不同，選擇針對這個平臺的 CrossGCC。

GCC 模式和 Java 模式的最大不同在於 GCC 直接生成目標平臺的目標代碼，而

Java 模式首先只是生成位元組碼，只有在有 JIT 編譯器的參與下才會進一步生

成目標平臺的目標代碼。研究表明，Java 模式雖然可以通過兩個編譯過程生成

目標代碼，但是因為兩次編譯的優化存在相互衝突，最終的目標代碼的執行效

率也不是很高。

而 GCC 模式由於直接能夠生成目標代碼，其執行效率一般很高。

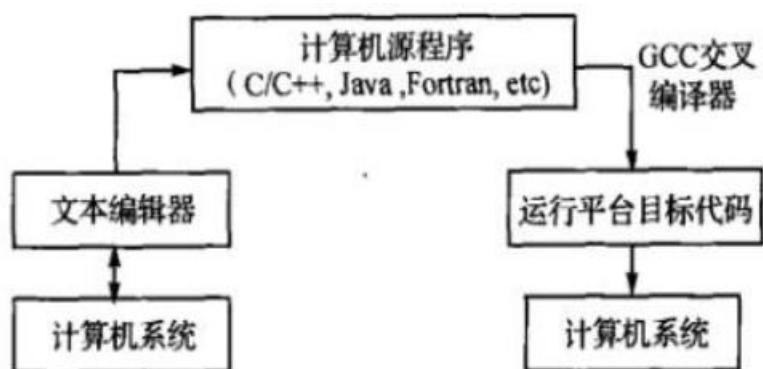


图 GCC 模式交叉编译

## 歷史:

早期的電腦軟體都是用**組合語言**直接編寫的，這種狀況持續了數年。當人們發現為不同類型的**中央處理器**（CPU）編寫可重用軟體的開銷要明顯高於編寫編譯器時，人們發明了高階程式語言。由於早期的電腦的**記憶體**很少，當大家實現編譯器時，遇到了許多技術難題。

大約在 20 世紀 50 年代末期，與機器無關的程式語言被首次提出。隨後，人們開發了幾種實驗性質的編譯器。第一個編譯器是由美國女性電腦科學家葛麗絲·霍普（Grace Murray Hopper）於 1952 年為 A-0 系統 編寫的。但是 1957 年由任職於 IBM 的美國電腦科學家約翰·巴科斯（John Warner Backus）領導的 FORTRAN 則是第一個被實作出具備完整功能的編譯器。1960 年，COBOL 成為一種較早的能在多種架構下被編譯的語言。

高階語言在許多領域流行起來。由於新的程式語言支援的功能越來越多，電腦的架構越來越複雜，這使得編譯器也越來越複雜。

早期的編譯器是用組合語言編寫的。首個能編譯自己源程式的編譯器是在 1962 年由麻省理工學院的 Hart 和 Levin 製作的。從 20 世紀 70 年代起，實現能編譯自己源程式的編譯器變得越來越可行，不過還是用 Pascal 和 C 語言 來實現編譯器更加流行。製作某種語言的第一個能編譯器，要麼需要用其它語言來編寫，要麼就像 Hart 和 Levin 製作 Lisp 編譯器那樣，用直譯器來執行編譯器。

## 在 Windows 上建立 C 語言開發環境

### 前言:

比起 Unix 或類 Unix 系統，Windows 對於 C 語言的原生支援較薄弱，缺乏單一最佳方案。目前來說，有以下四種方案：

Visual C++

Cygwin

WSL (Windows Subsystem for Linux) (Windows 10 限定)

MinGW + MSYS

選擇 C 開發環境時，會從 C 編譯器、編輯器、專案管理軟體、套件管理軟體等方面來思考。

## C 編譯器

C 編譯器包括編譯器本身和標準函式庫兩部分。

這兩個部分會包在一起，安裝 C 編譯器時即一併安裝。

## Visual C++

Visual C++ 是隨附在 Visual Studio 內的 C 和 C++ 編譯器。

安裝 Visual Studio Community 時，一併安裝 C++ 開發環境即可。

安裝完後，會得到 C 和 C++ 編譯器及一套完整的 IDE (整合式開發環境)，可以在完全不使用命令列工具的前提下學 C 語言，所以相當受歡迎。

然而，Visual C++ 在 C 標準上落後，而且微軟看起來沒有要更新該編譯器的 C 標準。如果有心要學現代 C 語言，或是注重 C 標準的版本差異的話，不建議使用 Visual C++ 來學。

但 Visual Studio 相當普遍，如果只是要學 C 基本語法，繼續使用 Visual C++ 無妨。

如果只是想用 Visual C++ 編譯 C 程式，但是不需要 Visual Studio 的 IDE 的話，可以考慮下載 Build Tools for Visual Studio。

下載後，同樣可以按照自己的需求選擇所需的元件，就和使用 Visual Studio 的安裝器雷同。

當我們安裝 Build Tools for Visual Studio 後，由於沒有 IDE 可用，得自行搭配其他的 IDE。

常見的選項有 **Code::Blocks** 或 **KDevelop** 等。

微軟希望大家直接去用 Visual Studio，所以對 Build Tools for Visual Studio 不太去宣傳，只是低調地放個連結給人下載。

但微軟也沒把這些連結拿掉，代表這個工具還是有其需求。

自己的推測，Build Tools for Visual Studio 主要是用在 Docker for Windows 這類純命令列環境的情境，而不是給一般開發者使用。



## Cygwin 內的 GCC

Cygwin 是一套運行在 Windows 上的類 Unix 子系統，主要的用途是在 Windows 上使用類 Unix 系統的命令列環境、命令列工具、開發工具等。為了要讓 Cygwin 支援 POSIX，Cygwin 附帶了一隻額外的 DLL (動態連結函式庫)。故在 Cygwin 下編譯的程式皆會依賴該 DLL。

我們會把 Cygwin 視為非 Windows 非 GNU/Linux 的特殊子系統。

如果只是想拿內附的 GCC 等開發工具來練習 C 語言倒是無妨。

如果想要開發 Windows 原生程式的話，還是使用其他的方案比較好。

# Windows Subsystem for Linux 內的 GCC

[WSL \(Windows Subsystem for Linux\)](#) 是一套運行在 Windows 上的

GNU/Linux 子系統，主要目的是提供類 Unix 系統上的開發工具。

原本的 WSL 是以 Linux 相容介面來運行，內部沒有真正的 Linux 核心。在

WSL2 後，會在該子系統內放入真正的 Linux 核心，利用虛擬化技術來運行。

由於 WSL 是相對新穎的軟體專案，該專案仍在演進中。有時候會在 WSL 上

碰到在原生類 Unix 系統上沒有的問題，這些問題可能難以解決或無法解決。

故現階段不建議使用 WSL 來練習 C 語言。

# MinGW + MSYS

MinGW 是 GCC 在 Windows 上的移植品，提供 C 和 C++ 編譯器、C 和 C++ 標準函式庫、開發工具等。但沒有第三方函式庫也沒有編輯器，這些部分要另外安裝。

至於 MSYS 則是隨附在 MinGW 內的小型類 Unix 子系統，主要用來移植來自類 Unix 系統的自由軟體。

原本的 MinGW 較不親民，後來有開發團隊製作 MSYS2。相較於 MinGW，MSYS2 除了提供 C 開發環境外，還加入套件管理軟體，要安裝第三方函式庫就方便得多了。但是編輯器的部分仍需另外安裝。

## 編輯器或 IDE

編輯器或 IDE 是用來撰寫程式碼的軟體。兩者的差別在於前者功能較單純，專注在撰寫程式碼本身；後者則加入較多功能，像是除錯、重構、專案管理、版本控制等。其實編輯器或 IDE 和 C 編譯器是脫勾的，但 Visual Studio 是少數將兩者同捆的例外之一。

## Visual Studio

Visual Studio 是微軟官方的 C 和 C++ IDE。由於裝好 IDE 後就可以取得完整的開發環境，故很受歡迎。

但 Visual Studio 只能搭配自家的 Visual C++，無法選擇其他的 C 和 C++ 編譯器。原本 Visual Studio 也只使用自家的 MSBuild 來管理專案，後來採取較開放的態度，開始接受 CMake 組態的專案。

## CLions

CLions 是一套跨平台的 C 和 C++ IDE。該 IDE 是商業軟體，但提供 30 天試用，可充份體驗後再決定是否要付費。CLions 沒有內建的 C 編譯器，但主流的 C 編譯器均有支援。除了 C 和 C++，CLions 還支援 Objective-C、Python、網頁前端等多種語言。

為了能夠在不同平台下管理專案，CLions 採用 CMake。雖然有一些 CLions 使用者希望 CLions 能夠加入 Makefile 的支援，但 CLions 目前不支援 Makefile。可能的原因是 Make 程式在不同平台間的歧異較大。

## Code::Blocks

Code::Blocks 是一套跨平台的 C、C++、Fortran IDE，不僅開放原始碼還可免費取得。該 IDE 沒有內建的 C 編譯器，可和多種 C 編譯器搭配，包括 MinGW。然而，Code::Blocks 不接受 CMake 或 Make 組態的專案，而是使用自家的專案管理程式。

## KDevelop

KDevelop 是一套基於 Qt 的跨平台 C 和 C++ IDE，本身是可免費取得的自由軟體。該 IDE 沒有內建的 C 編譯器，可搭配多種 C 編譯器，包括 Visual C++ 及 MinGW。雖然 KDevelop 的名氣不若前述各 IDE 響亮，此 IDE 支援 CMake、QMake、Make 等多種組態的專案，相當方便。

## Dev-C++

Dev-C++ 是一套免費的 C 或 C++ IDE，在台灣的大專院校間很流行。但我們不應再使用 Dev-C++。

Dev-C++ 會紅的原因是早期的 Visual Studio 很貴，免費的 Express 版本功能限制多，所以大家會轉而使用 Dev-C++。

但 Dev-C++ 本身已經不再維護了，而且 Visual Studio Community 的功能也相當完整，沒有理由繼續使用 Dev-C++。

## 其他編輯器

大部分編輯器對 C 都有基本的支援。

對於小型檔案，不一定要開 IDE 來用。以下是一些常見的編輯器：

Atom

Sublime Text

Visual Studio Code ([VSCode](#))

[Notepad++](#) (Windows 限定)

Vim

Emacs

在這些編輯器中，[VSCode](#) 是近年來竄升最快的編輯器。

該編輯器不僅跨平台，有著豐富的外掛，還支援多種語言，是輕量編輯的最佳選擇。

## 管理 C 專案

C 語言本身沒有專案的概念，由各個系統自行提供專案管理方案。常見的有以下數種方案：

使用 IDE 內建的專案管理軟體

使用 Make

使用 CMake

使用其他專案管理軟體

對於初學者來說，當下的學習重點在於學會 C 的核心功能，不應耗費過多時間在專案組態上。所以直接使用 Visual Studio 或 Code::Blocks 這類內建專案管理功能的 IDE 無妨。

若是要對外發佈的專案，則不應綁死特定 IDE，這時候應該用 Make、CMake 等不綁定 IDE 的專案管理工具。直接使用 Visual Studio 來管理專案的前提是該專案只會在 Windows 上發佈。



## 安裝第三方函式庫

在程式寫到一定規模後，標準函式庫會不敷使用，這時候就會尋求適合的第三方函式庫。C 語言有函式庫的格式但沒有套件的概念。

在 Windows 上也沒有 `/usr/include` 或 `/usr/lib` 等系統內定函式庫路徑，而是用手動的方式將函式庫的標頭檔及二進位檔拷貝到專案中。

但純手動不易管理，所以發展出下列安裝及管理第三方套件的方式：

使用 `vcpkg`

使用 MSYS2 內附的 `Pacman`

`vcpkg` 是微軟自家的第三方套件管理軟體。該專案為許多受歡迎的 C 或 C++ 函式庫寫 CMake 組態，再利用 `vcpkg` 主程式來自動安裝這些函式庫。但目前的 `vcpkg` 無法自行設置套件的編譯參數，而且只能搭配 Visual C++。

`Pacman` 原本是 Arch Linux 的套件管理軟體，MSYS2 專案將其移植到 Windows 上，用來搭配 MinGW。同樣地，MSYS2 專案預先將許多受歡迎的 C 或 C++ 函式庫包成套件，所以可以自動安裝。

為什麼會有兩套重覆功能的套件管理軟體？這是為了搭配不同的 C 編譯器所致。

## 如何選擇？

由於 Cygwin 及 WSL 需在子系統中運行，故先排除。實際在選擇時，就是在 Visual C++ (MSVC) 及 MinGW (GCC) 兩大 C 編譯器系統擇一。

如果程式只要在 Windows 上跑，可以直接用 Visual Studio 全家餐，就會有完整的 C 編譯器 (Visual C++) 及 IDE。頂多在需要第三方函式庫時拉 vcpkg 來用。

如果程式需要在多個平台上運行，就應該用 MinGW (GCC)。這時候的組合會是 MSYS2 + Make (或 CMake) + 第三方編輯器 (或 IDE)。

對於初學者來說，Visual Studio 或 Code::Blocks 這類內建專案管理功能的 IDE 比較方便。但在跨過新手階段後，應該要試著用 CLions 或 KDevelop 這類可跨平台的 IDE，並用 CMake 或 Make 管理專案。

## 使用 MinGW (MSYS2) 搭配 Code::Blocks

綜合以上的想法，建議以下的組合（擇一即可）：

MSYS2 搭配 Code::Blocks

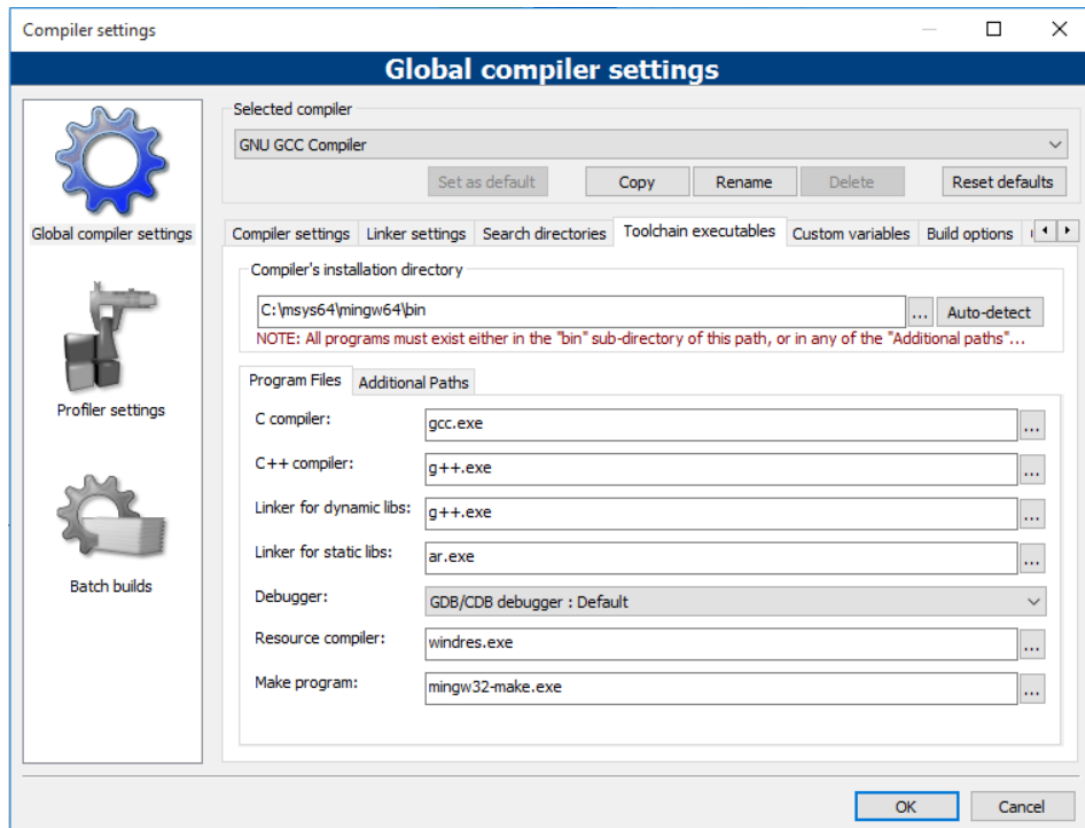
MSYS2 搭配 Visual Studio Code

Code::Blocks 的好處在於本身為跨平台軟體，可在 Windows、GNU/Linux、Mac 下執行；此外，Code::Blocks 可選用內建的事案管理程式或 Makefile，對初學者來說，可先用內建的事案管理系統減輕學習的負擔，上手後再慢慢轉到 Makefile。如果要用 Visual Studio Code，建議學一些基本的 Makefile 撰寫方式，簡化編譯軟體的過程。

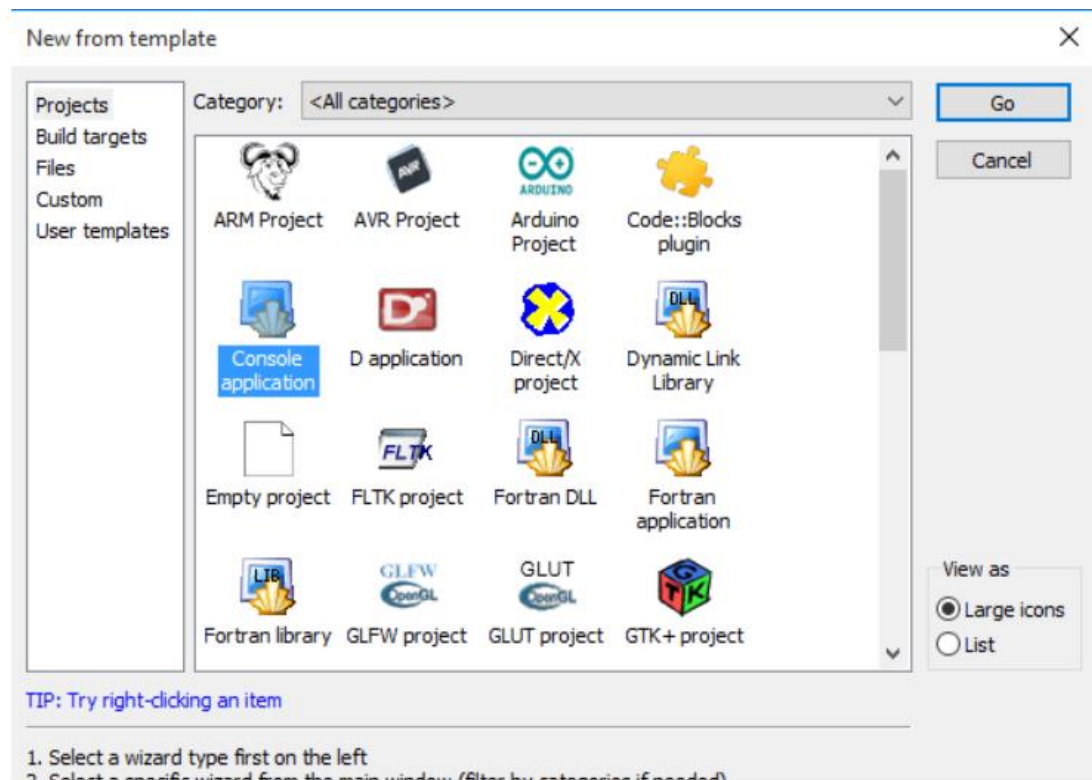
接下來，我們用經典的 Hello World 範例來說明如何在 Windows 下撰寫 C 語言。使用 IDE 的讀者，請建立一個終端機程式的事案，使用編輯器的讀者，可建立 hello.c 檔案（也可用其他名稱，建議用英文來命名）。我們這裡以 Code::Blocks 為例，來展示建立專案過程。

如果 Code::Blocks 無法抓到 GCC (MinGW)的路徑

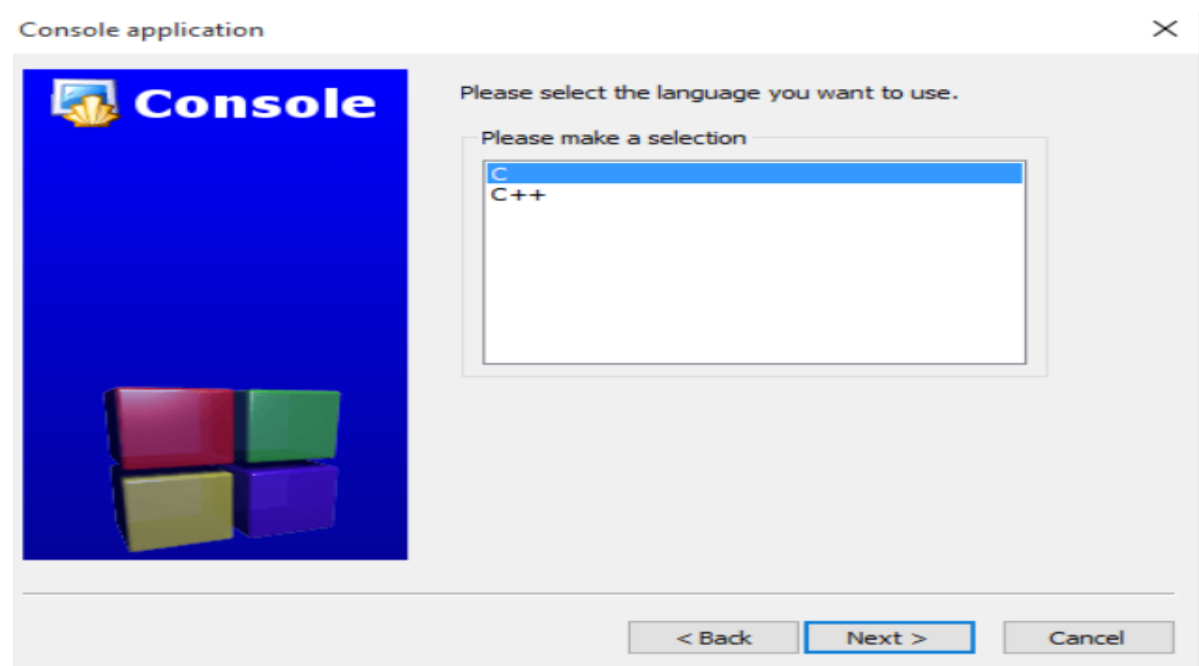
如果 Code::Blocks 無法抓到 GCC (MinGW) 的路徑，需手動修改  
GCC 路徑 (從 Settings 的 Compiler 選單選擇)：



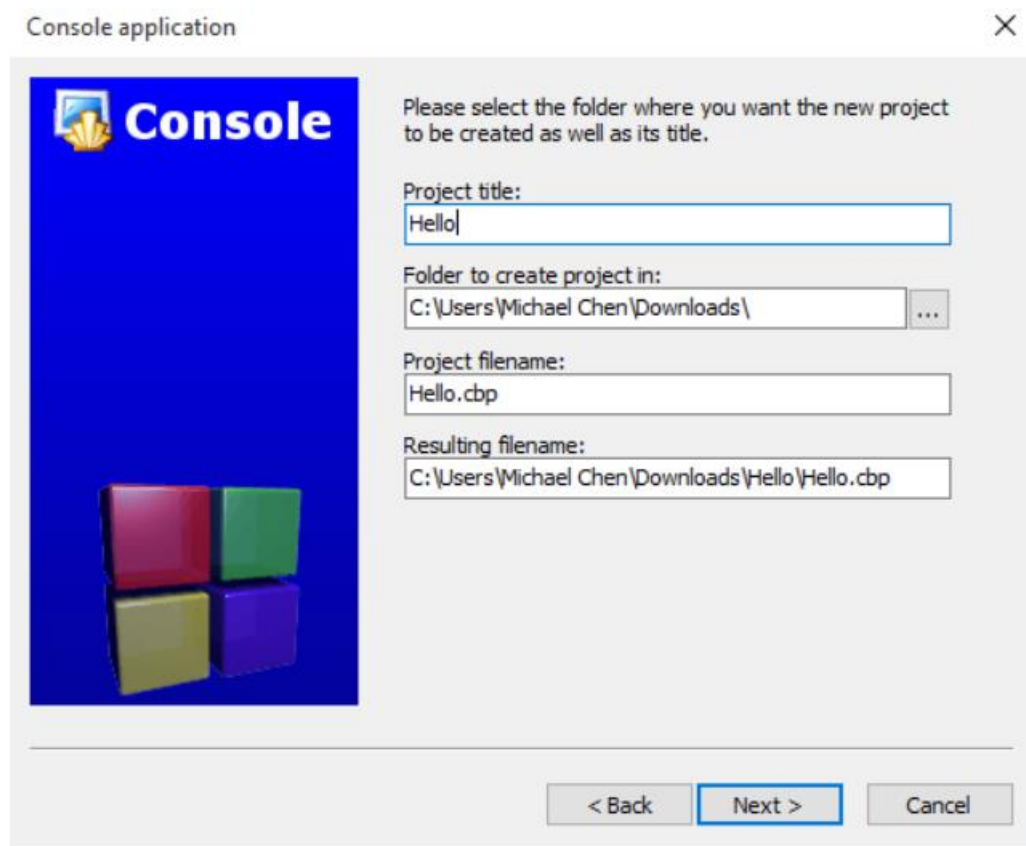
選擇專案類型，這裡選「Console application」：



選擇專案語言，記要選 C，不要選 C++，因為 C++ 並不是 C 的超集合 (superset)：



設置專案名稱和路徑：



The screenshot shows the 'Console application' wizard window. On the left is a blue sidebar with the 'Console' logo and a 3D cube graphic. The main area has a light gray background with the instruction: 'Please select the folder where you want the new project to be created as well as its title.' Below this are four text input fields: 'Project title:' with 'Hello', 'Folder to create project in:' with 'C:\Users\Michael Chen\Downloads\' and a browse button (...), 'Project filename:' with 'Hello.cbp', and 'Resulting filename:' with 'C:\Users\Michael Chen\Downloads\Hello\Hello.cbp'. At the bottom are three buttons: '< Back', 'Next >', and 'Cancel'.

Console application

Please select the folder where you want the new project to be created as well as its title.

Project title:  
Hello

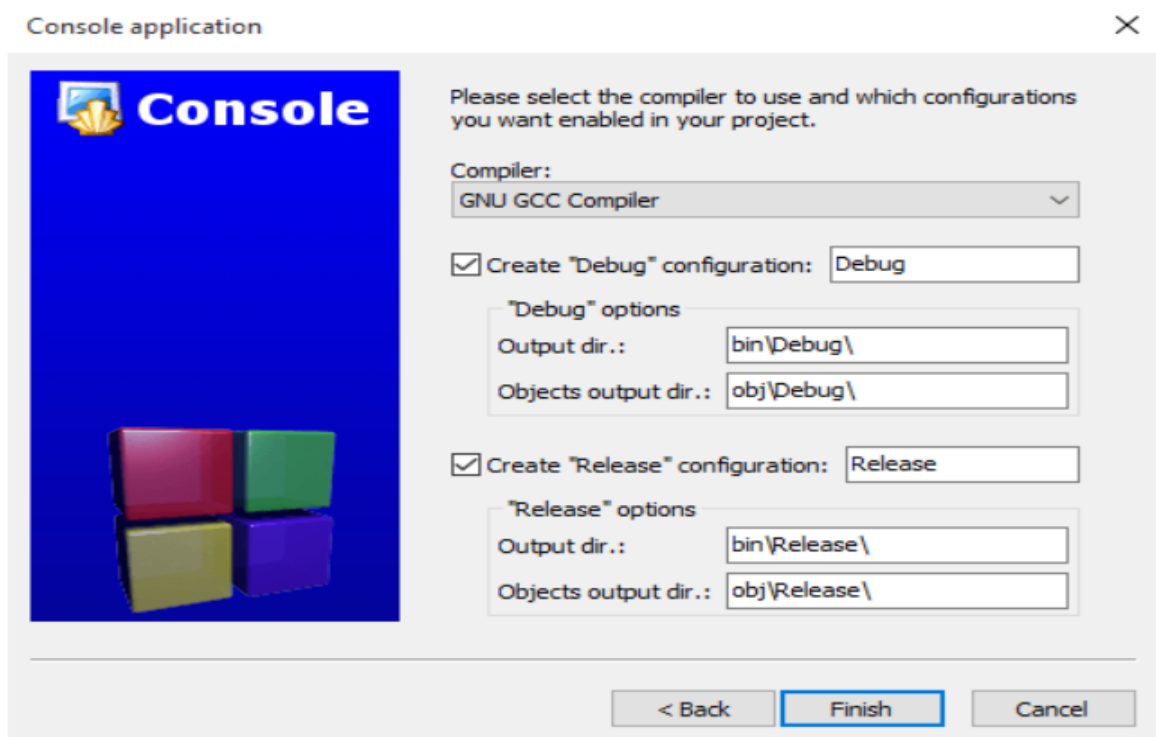
Folder to create project in:  
C:\Users\Michael Chen\Downloads\ ...

Project filename:  
Hello.cbp

Resulting filename:  
C:\Users\Michael Chen\Downloads\Hello\Hello.cbp

< Back   Next >   Cancel

選擇編譯器，這裡選 GCC 即可：



The screenshot shows the second step of the 'Console application' wizard. The left sidebar is the same. The main area has the instruction: 'Please select the compiler to use and which configurations you want enabled in your project.' Below this are several settings: a 'Compiler:' dropdown menu set to 'GNU GCC Compiler'; a checked checkbox 'Create "Debug" configuration:' with a 'Debug' text box; a section for 'Debug' options with 'Output dir.:' set to 'bin\Debug\' and 'Objects output dir.:' set to 'obj\Debug\'; another checked checkbox 'Create "Release" configuration:' with a 'Release' text box; and a section for 'Release' options with 'Output dir.:' set to 'bin\Release\' and 'Objects output dir.:' set to 'obj\Release\'. At the bottom are three buttons: '< Back', 'Finish', and 'Cancel'.

Console application

Please select the compiler to use and which configurations you want enabled in your project.

Compiler:  
GNU GCC Compiler

☒ Create "Debug" configuration: Debug

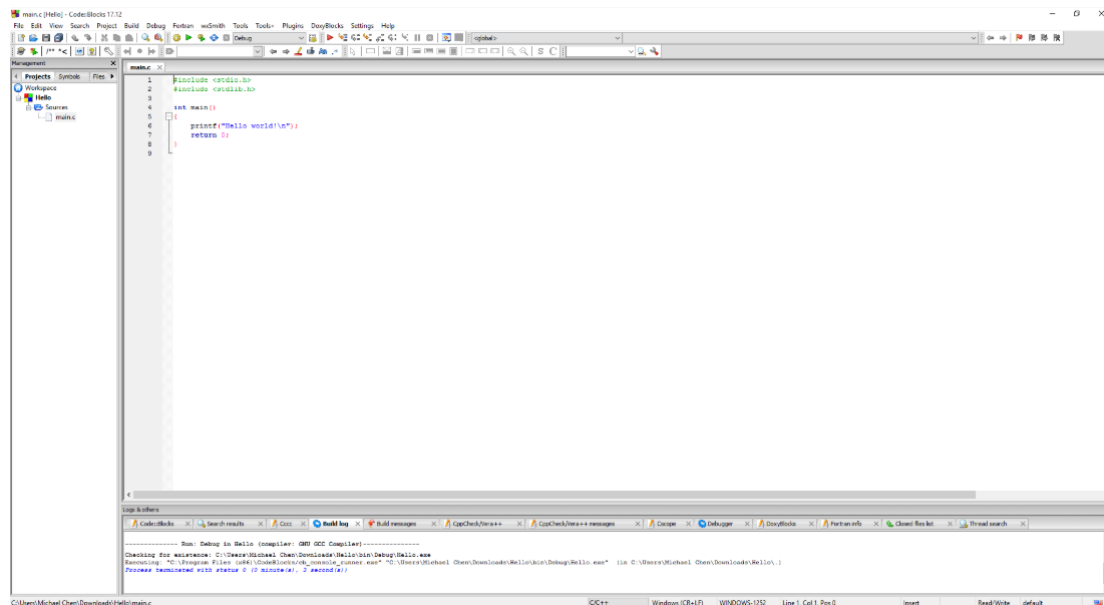
"Debug" options  
Output dir.: bin\Debug\  
Objects output dir.: obj\Debug\

☒ Create "Release" configuration: Release

"Release" options  
Output dir.: bin\Release\  
Objects output dir.: obj\Release\

< Back   Finish   Cancel

進入 Code::Blocks 的編輯器，開始撰寫程式：



撰寫第一個程式

我們這裡展示 Hello World 程式，暫時不要管程式碼的意義，這裡的重點是確保程式可順利運行：

```
#include <stdio.h>

int main()
{
    printf("Hello World\n");

    return 0;
}
```

# 你所不知道的 C 語言：編譯器原理和案例分析

如何打造一個具體而微的 C 語言編譯器？

700 行系列 與隨感

整整 20 年前，我在苗栗老家用著緩慢的 Pentium 電腦搭配硬碟，在 Slackware Linux 用舊版 gcc 編譯和安裝 gcc-2.8.0(1998 年 1 月 14 日釋出) 和 gcc-2.8.1(同年 3 月釋出)。

gcc 8.2.0 版於 2018 年 7 月 26 日釋出

你可以想像今日是工業編譯器技術標準的 gcc，在整整 20 年前，原始碼打包後僅 8MB，全部程式碼不到 53 萬行 (C 語言佔其中近 33 萬行)，而且測試程式碼才 241 行嗎？對比最新的 gcc-8.2.0，用 tar + gzip 壓縮打包後，後者佔了 109MB，原始程式碼的規模已達到約 752 萬行

20 年的時間，一個專案從「免費又好用」變成「帶領產業變革的標準」，原始程式碼膨脹 14 倍 (過程中移除 gcj 一類的子專案，實際上要大得多)

gcc-2.8.0 程式碼中，可見到 Richard Stallman 修改程式碼的蹤跡 (參見 ChangeLog 檔案)。

AMaCC 是由台灣國立成功大學師生開發的 self-compiling 的 C 語言編譯器，可產生 Arm 架構的執行檔 (ELF 格式，運作於 GNU/Linux)、也支援 just-in-time (JIT) 編譯和執行，原始程式碼還不到 1500 行，在本次講座中，我們揭開 AMaCC 背後的原理和實作議題。

預期會接觸到 IR (Intermediate representation), dynamic linking, relocation, symbol table, parsing tree, language frontend, Arm 指令編碼和 ABI 等等。



## 編譯器和軟體工業強度息息相關

依據 IBM 的簡報指出: [ source ]

1960 年代的 F-4 Phantom 戰鬥機: 僅有 8% 需要軟體控制 (組合語言開發);

1970 年代的 F-16 Falcon 戰鬥機: 45% 需要軟體控制 (JOVIAL 程式語言開發, 類似 ALGOL, 主要針對嵌入式系統用於編寫軍用飛機電子系統);

1980 年代的 F-22 Raptor 戰鬥機: 80% 規範仰賴軟體 (Ada83 程式語言開發), 程式碼規模為 170 萬行;

2000 年之後的 F-35 Lightning 戰鬥機: 2400 萬行程式碼, 用 C/C++ 撰寫;  
自動駕駛車的軟體規模預計會百倍於飛行控制系統, 而開發階段所必須投入發展的軟體驗證環境和方法對開發都至關重大。

韓國的 Hyundai Mobis 計畫與 KAIST 合作開發 MAIST, 著眼於後續自駕車的軟體性能及成本的影響非常龐大。

IEEE 1012: 安全相關應用軟體開發

IEC 61508-3: 以每小時危險失效率 (PFH) 或失效危險機率 (PFD) 的形式訂立許多可靠度要求形式化驗證。

## 參考資料

Lua 教學 : [http://tw.gitbook.net/lua/lua\\_environment.html](http://tw.gitbook.net/lua/lua_environment.html)

編譯器-MBA 智能百科:

<https://wiki.mbalib.com/zh-tw/%E7%BC%96%E8%AF%91%E5%99%A8>

Compiler 介紹:

<https://lagunawang.pixnet.net/blog/post/9743288compiler%e4%bb%8b%e7%b4%b9>

[C 語言] 程式設計教學：在 Windows 上建立 C 語言開發環境

<https://michaelchen.tech/c-programming/write-c-on-windows/>

你所不知道的 C 語言：編譯器原理和案例分析

<https://hackmd.io/@sysprog/c-compiler-construction?type=view>

谢谢观看

