



CHANGED A MONTH AGO



OWNED THIS NOTE



Edit

你所不知道的 C 語言：開發工具和規格標準

Copyright (憲C) 2015, 2017 宅色夫

直播錄影

“If I had eight hours to chop down a tree, I'd spend six hours sharpening my axe.” – Abraham Lincoln

「如果我有 8 小時可以砍 1 棵樹，我會花 6 小時把斧頭磨利。」(類似漢語「工欲善其事，必先利其器」的精神) – 亞伯拉罕. 林肯

語言規格：

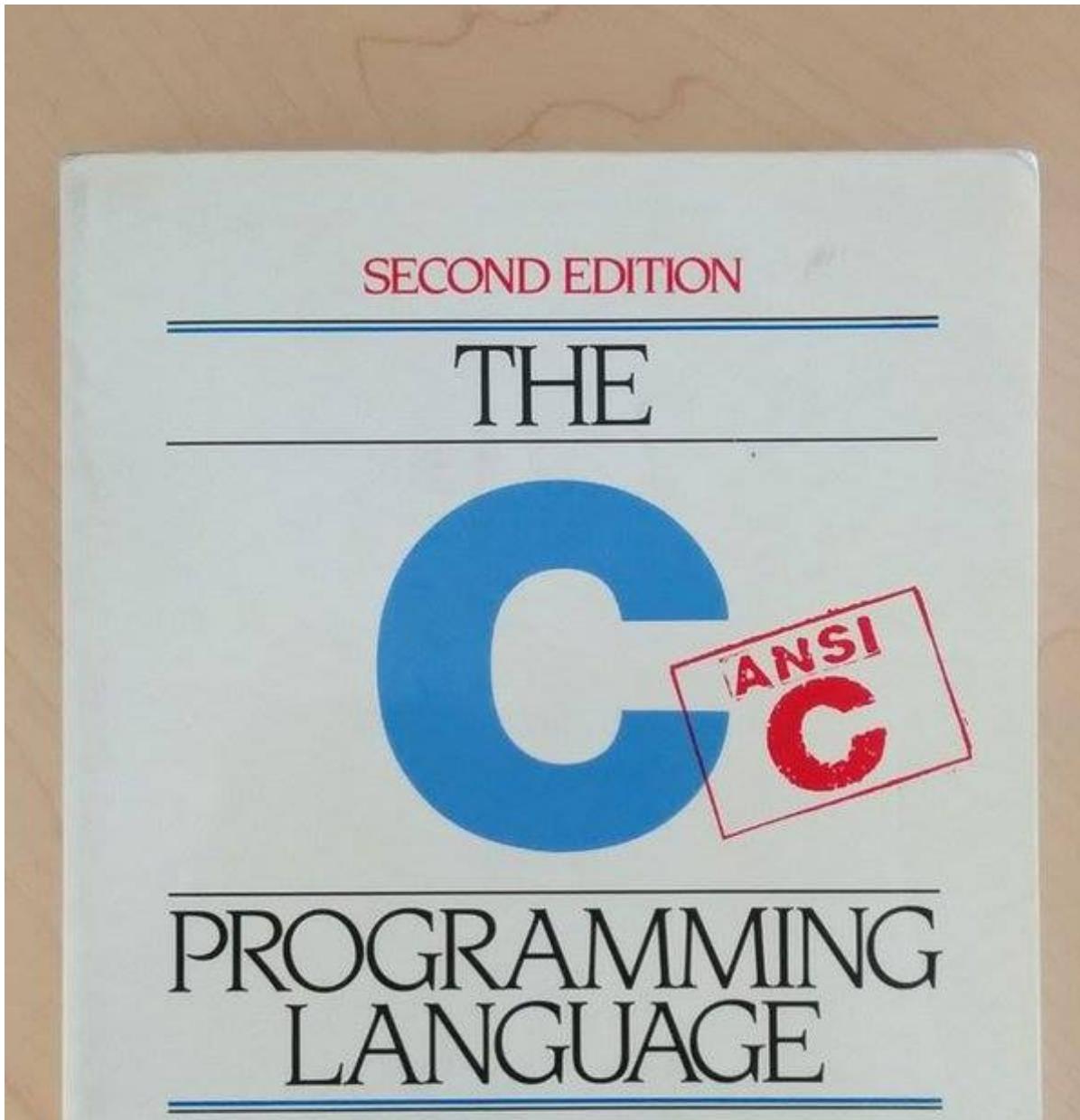
C89/C90 -> C99 -> C11 -> C17/C18 -> C2x

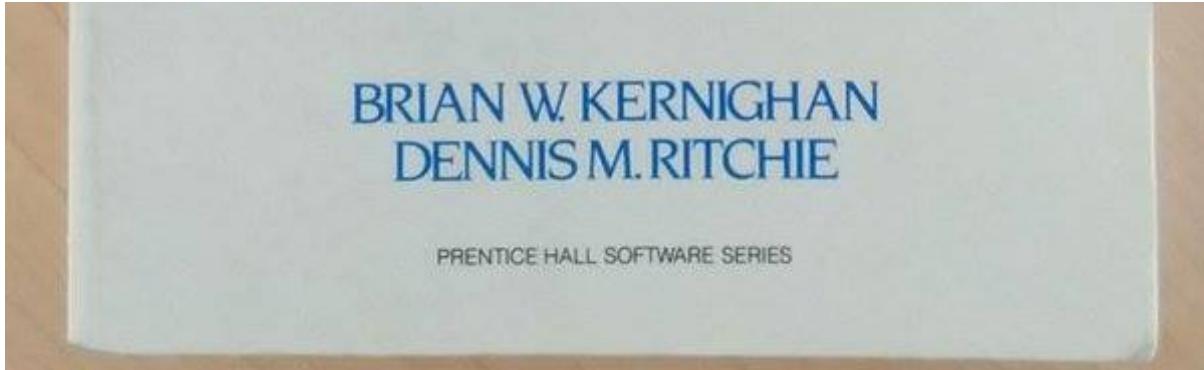
C 語言老爸的評論

「C 很彆扭又缺陷重重，卻異常成功。固然有歷史的巧合推波助瀾，可也的確是因為它能滿足於系統軟體實作的程式語言期待：既有相當的效率來取代組合語言，又可充分達到抽象且流暢，能用於描述在多樣環境的演算法。」

C is quirky, flawed, and an enormous success. Although accidents of history surely helped, it evidently satisfied a need for a system implementation language efficient enough to displace

assembly language, yet sufficiently abstract and fluent to describe algorithms and interactions in a wide variety of environments. —— Dennis M. Ritchie





Jonathan Adamczewski 貼出經典著作《The C Programming Language》，然後評註說：

“C++: The Good Parts”

K&R C 的 “K” 即 Brian Kernighan 教授，而非 Ken Thompson。



David Brailsford 教授的訪談，從大型主機 (mainframe) 時代談起，早期的機器甚至不是 byte addressing，這使得高階程式語言和電腦硬體無法直接對應，而 C 語言恰好彌補系統層級程式設計的需求，再藉由 UNIX 作業系統的影響，改變今日我們所處的資訊世界。

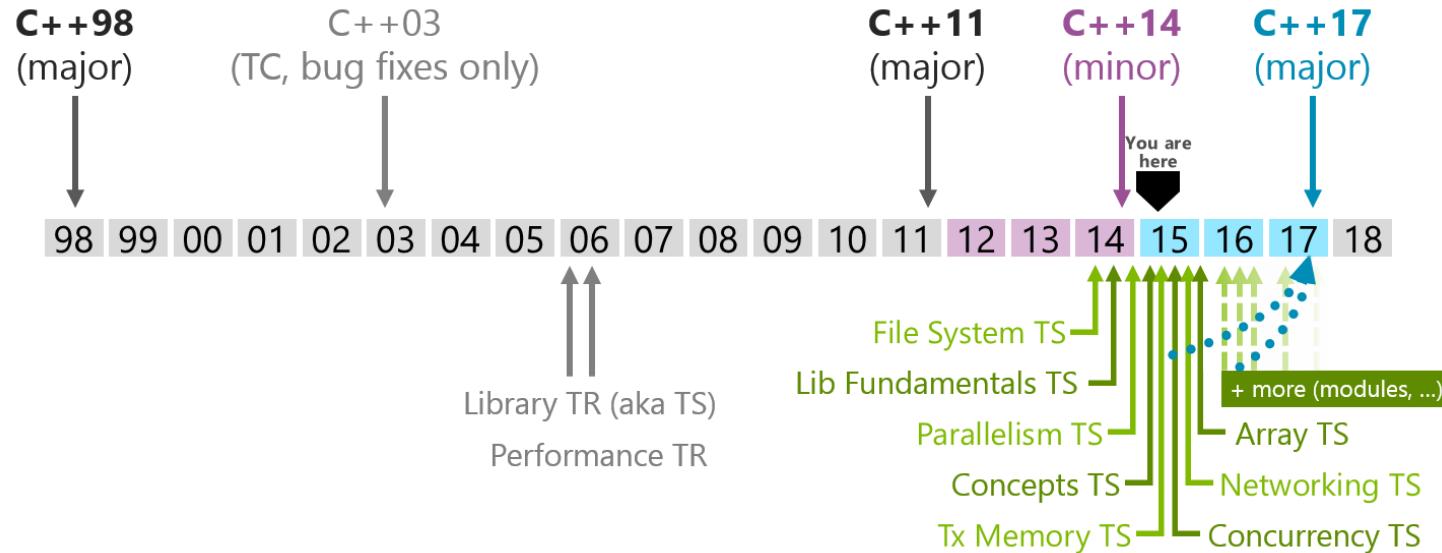
c++andy c++rush

```
int main()
{
    []() {};
    []{}();
    {}[]{};
}
```

C++ 可以美得令人不知所措 [\[source\]](#)

為什麼我不探討 C++

- 在台灣發文好像愛用「為什麼我不」開頭，後面可接「念研究所」、「待在大公司」等描述
- C++ 自稱為物件導向的程式語言，卻不願意對物件在執行時期的表現負責任
 - 若說 C 語言給了你足夠的繩子吊死自己，那麼 C++ 紿的繩子除了夠你上吊之外，還夠綁住你身邊的朋友
 - 相較之下，Java 讓你在吊死自己之際仍有親友監視著，雖然死不了，但事後會更想死
 - [[source](#)]
 - In Ruby, everything is an object.
 - In Clojure, everything is a list.
 - In Javascript, everything is a terrible mistake.
 - in C, everything is a representation (unsigned char [sizeof(TYPE)]).
- Linus Torvalds 在 [2010 年的解釋](#)
- C++ 實際上已經是截然不同的程式語言
 - C++ 老爸 Bjarne Stroustrup 的文章: “[Learning Standard C++ as a New Language](#)”
- 最重要的是，C++ 改版飛快，C++ 17 即將推出，但我還沒看懂 C++ 98



- [[source](#)]

延伸閱讀

- 沒有 C 語言之父，就沒有 Steve Jobs ([原文](#))
- 第一個 C 語言編譯器是怎樣編寫的？



讀規格書可大幅省去臆測

在 ISO/IEC 9899 (a.k.a C99 Standard) 中 5.1.2.2.1 內有提到 C Standard 要求 main 函數必須這樣寫

```
int main(void) { /* ... */};
```

或者:

```
int main(int argc, char *argv[]) { /* ... */};
```

C++ 之父 Bjarne Stroustrup 的個人網頁內有個 [FAQ](#) 裡面有個問題叫 **Can I write “void main()”?**

然而在 C++ 與 C 的標準中從來沒出現過這樣的寫法，也就是說，`void main()` 這個寫法從來沒正確過

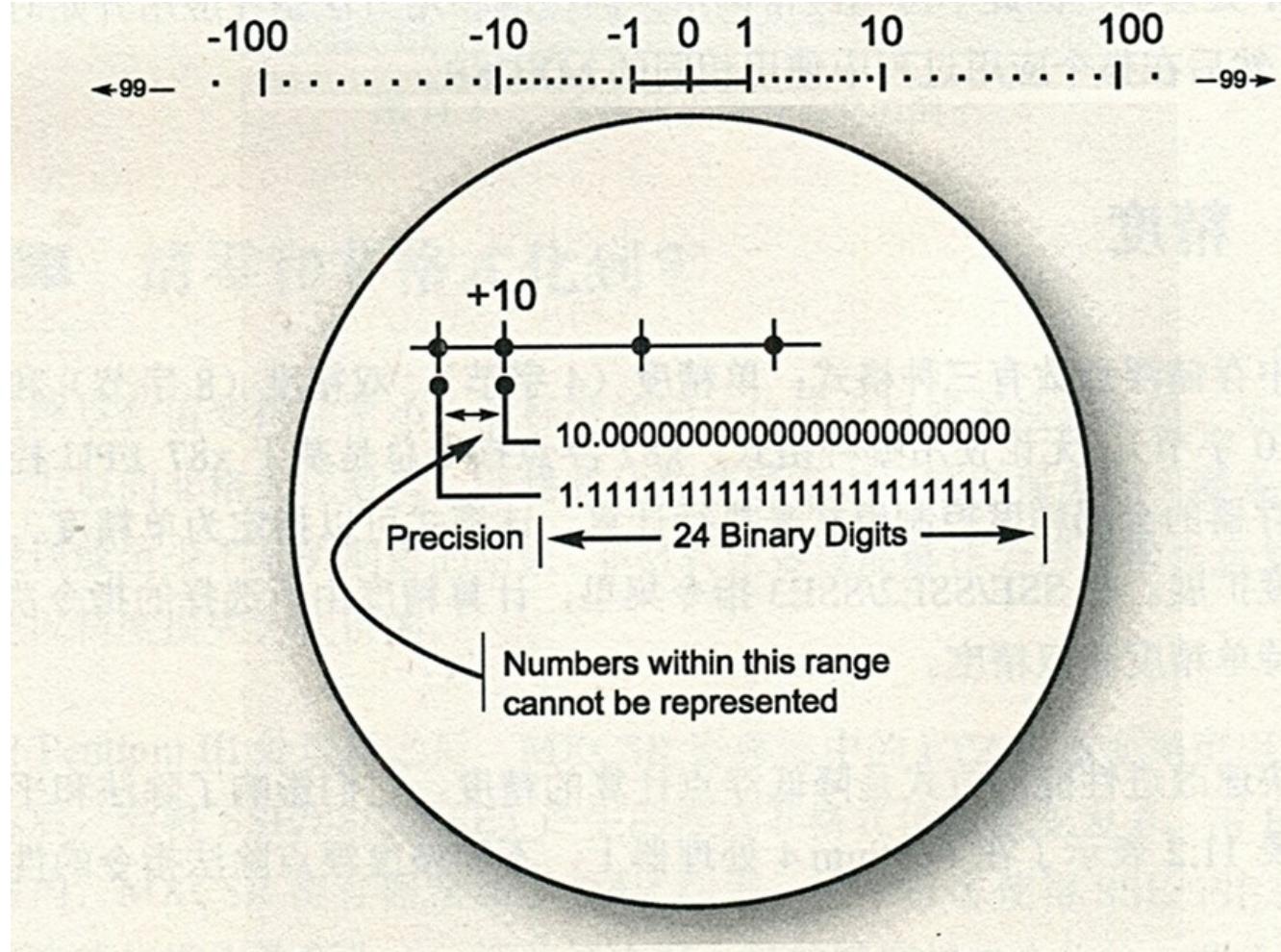
延伸閱讀

- C 語言中 int main() 和 void main() 有何區別？
 - C++ 的 void main() / int main() ... 不要再用 void main() 了! | Peter Dave Hello's Blog
 - void main(void) - the Wrong Thing
-

ISO/IEC 9899 (簡稱 “C99”)

- 從[一則笑話](#)談起
 - “Programming in C: if it doesn’t work, just add a star. Or multiple stars. Or ampersands.”
- 葉秉哲博士的[推文](#)：「[溯源能力](#)是很重要的，才不會被狀似革新，實則舊瓶裝新酒或跨領域借用的『新觀念』所迷惑」
- 規格書 (PDF) 搜尋 “**object**”，共出現 735 處
 - 搜尋 “**pointer**”，共出現 637 處。有趣的是，許多教材往往不談 object，而是急著談論 pointer，殊不知，這兩者其實就是一體兩面
 - object != object-oriented
 - 前者的重點在於「資料表達法」，後者的重點在於“everything is object”
 - C11 ([ISO/IEC 9899:201x](#)) / [網頁版](#)
- 從第一手資料學習：大文豪寫作都不免要查字典，庸俗的軟體開發者如我們，難道不需要翻閱語言規格書嗎？難道不需要搞懂術語定義和規範嗎？

- & 不要都念成 and，涉及指標操作的時候，要讀為 “address of”
 - C99 標準 [6.5.3.2] Address and indirection operators 提到 ‘&’ address-of operator
- C99 [3.14] **object**
 - region of data storage in the execution environment, the contents of which can represent values
 - 在 C 語言的物件就指在執行時期，資料儲存的區域，可以明確表示數值的內容
 - 很多人誤認在 C 語言程式中，(int) 7 和 (float) 7.0 是等價的，其實以資料表示的角度來看，這兩者截然不同，前者對應到二進位的“111”，而後者以 IEEE 754 表示則大異於“111”



- A pointer to void shall have the same representation and alignment requirements as a pointer to a character type.

關鍵描述！規範 `void *` 和 `char *` 彼此可互換的表示法

```
void *memcpy(void *dest, const void *src, size_t n);
```

- C99 規格書的解說就比很多書本清楚，何必捨近求遠呢？
 - **EXAMPLE 1:** The type designated as `float *` has type “pointer to float”. Its type category is pointer, not a floating type. The const-qualified version of this type is designated as `float * const` whereas the type designated as `" const float *` is not a qualified type — its type is “pointer to const qualified float” and is a pointer to a qualified type.
 - **EXAMPLE 2:** The type designated as `" struct tag (*[5])(float)` has type “array of pointer to function returning struct tag”. The array has length five and the function has a single parameter of type float. Its type category is array.
- [Understand more about C](#) 提及若干肇因於不同的 C 語言標準，而使得程式碼行為不同的案例

規格不能只看新的，過往也要熟悉



source

- 空中巴士 330 客機的娛樂系統裡頭執行 14 年前的 Red Hat Linux，總有人要為「古董」負責
- 而且空中巴士 380 客機[也是如此](#)

為何 C 語言標準函式庫裡頭的函式名稱如此簡短？像是

- strcpy
- strlen

最初連結器有 6 到 8 個字元的輸入限制!

Translation limits

6 significant initial characters in an external identifier

- 延伸閱讀:

- [Why did ANSI only specify six characters for the minimum number of significant characters in an external identifier?](#)
 - [Identifier](#)

3.3.3 Linkage Problems

As mentioned, I decided to live within the constraints of traditional linkers. However, there was one constraint I found insufferable, yet so silly that I had a chance of fighting it if I had sufficient patience: Most traditional linkers had a very low limit on the number of characters that can be used in external names. A limit of eight characters was common, and six characters and one case only are guaranteed to work as external names in K&R C; ANSI/ISO C also accepts that limit. Given that the name of a member function includes the name of its class and that the type of an overloaded function has to be reflected in the linkage process somehow or other (see §11.3.1), I had little choice.

Consider:

```
void task::schedule() { /* ... */ } // 4+8 characters  
  
void hashed::print() { /* ... */ } // 6+5 characters  
  
complex sqrt(complex); // 4 character plus 'complex'  
double sqrt(double); // 4 character plus 'double'
```

Representing these names with only six upper case characters would require some form of compression that would complicate tool building. It would probably also involve some form of hashing so that a rudimentary ‘program database’ would be

英文很重要

安裝 `cdecl` 程式，可以幫你產生 C 程式的宣告。

```
$ sudo apt-get install cdecl
```

使用案例

```
$ cdecl  
cdecl> declare a as array of pointer to function returning pointer to function ret
```

會得到以下輸出：

```
char * (*(*a[]))()
```

把前述 C99 規格的描述帶入，可得：

```
cdecl> declare array of pointer to function returning struct tag
```

```
struct tag (*var[])()
```

如果你沒辦法用英文來解說 C 程式的宣告，通常表示你不理解！

`cdecl` 可以解釋 C 程式宣告的意義，比方說：

```
cdecl> explain char *(*fptab[]) (int)  
declare fptab as array of pointer to function (int) returning pointer to char
```

只用 printf 觀察資料，有問題嗎？

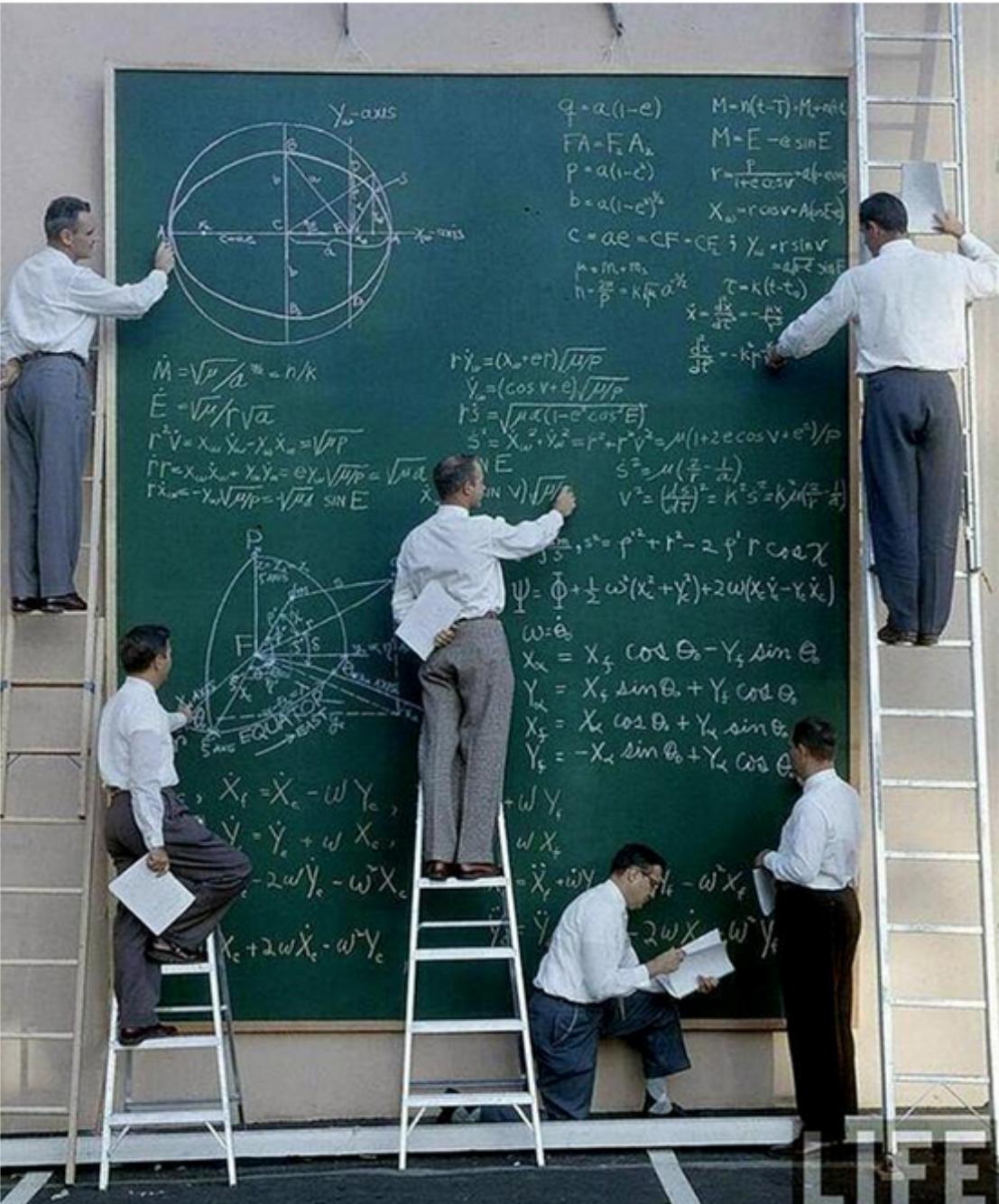




[[source](#)]

- 只用 `printf()` 觀察的話，永遠只看到你設定的框架 (format string) 以內的資料，但很容易就忽略資料是否合法、範圍是否正確，以及是否看對地方
- `printf()` 大概是最早被記下來的函式，也困擾很多人，有意思的是，1960 年代初期 MIT 開發的 [CTSS 作業系統](#) 中，終端機命令就包含了 `printf`，後者一路從 Multics 和 Unix 繼承至今
- 在 CTSS 原始程式碼的檔案 `com3` 中可見到這行 `STMTDC PRINTF,11,T,T25`，前一行註解寫
“The following tables are the dictionaries of statement types”

不要急著印出位址，善用 GDB



source: NASA before PowerPoint, 1961

- 「學會了 GDB，我有種山頂洞人學會用火的感動」 – 張至
- [GDB Rocks!](#) (on slideshare paid)
- [Introduction to gdb](#) (on slideshare paid)
- [Debugging with GDB](#) (on slideshare paid)
 - [clewn, pyclewn on sourceforge](#)
 - [gdb to kernel](#)
 - [GDB 的妙用](#)
 - [Kaie's Blog](#)
 - [GDB Documentation](#)
- 除錯程式: [gdb](#)
- [Introduction to GDB a tutorial - Harvard CS50](#) (教學影片)
- [透過 GDB 學習 C 語言](#)

GDB

- [Kernel command using Linux system calls](#)
- video: [Linux basic anti-debug](#)
- video: [C Programming, Disassembly, Debugging, Linux, GDB](#)
- [rr](#) (Record and Replay Framework)
 - video: [Quick demo](#)
 - video: [Record and replay debugging with “rr”](#)

除了 Vim，我推薦 Visual Studio Code

-
- Microsoft 開放原始碼專案 Visual Studio Code (VS Code)
 - video: [五個 Visual Studio Code 的實用工具與技巧](#)
 - [共筆](#)

C2x

C 語言規格一直在變革，此刻最新的標準是 C17，其正式名稱為ISO/IEC 9899:2018，是在 2017 年準備、隔年發布的規範。C2x 則是現行開發中規格的簡稱，預計在 2023 年進行投票，屆時預計會發佈新標準，可稱為 C23。預計納入的特徵如下：

- `typeof` 運算子，過往是 GNU extension，這可用來實作 `container_of` 一類的巨集
- 強制規範 `call_once` 的支援，後者在並行 (concurrent) 環境中，得以確保某段程式碼始終只會執行一次
- 納入 `char8_t` 的支援，以 `_t` 結尾的慣例說明這由編譯器供應商提供的 `typedef`，這對 Unicode 更友好，於是我們可以書寫為 `u8"❀"[0]`
- 提供 `unreachable()`，讓編譯器得以進行更激進的最佳化，過往是 GNU extension
- 以 `= {}` 取代 `memset` 函式的呼叫
- 支援 ISO/IEC 60559:2020，這是最新 IEEE 754 浮點數運算標準
- 針對 C11 紳入的 `_Static_assert`，允許單一參數
- 納入 C++11 風格的 `attribute` 語法，例如 `nodiscard`, `maybe_unused`, `deprecated`, 和 `fallthrough`
- 新的函式: `memccpy()`, `strdup()`, `strndup()` – 類似 POSIX/SVID 中 C 函式庫的擴充
- 強制規範使用二補數符號表示
- 不再支援 K&R 風格的函式定義

- 二進位表達式 (Binary literals)，例如 `0b10101010` (很難想像 1970 年代發展的 C 語言，要到 2023 年或稍晚，才能直接指定二進位表示吧？) 和對應 `printf()` 的 `%b`
- Type generic functions for performing checked integer arithmetic (Integer overflow)
- `_BitInt(N)` and `UnsignedBitInt(N)` types for bit-precise integers
- 支援 `#elifdef` 和 `#elifndef`
- 允許在數值表示中加上分隔符號，易於閱讀，例如 `0xFFFF'FFFF`

延伸閱讀: [Ever Closer - C23 Draws Nearer](#)

Published on  HackMD

 68608

 18

