程式語言一　講義

開發環境建置

Hanly《C 語言詳論》6e

§ 1　名詞速查表

中文	English	一句話定義	科目	章節
編輯器	editor	寫程式碼的工具（本課用 VS Code），像 Word 之於文件	程式語言一	開發環境建置
編譯器	compiler	把 C 原始碼翻成電腦看得懂的機器碼的工具（本課用 GCC）	程式語言一	開發環境建置
原始碼	source code	人看得懂的程式文字，如 `hello.c`	程式語言一	開發環境建置
機器碼	machine code	電腦真正執行的 0/1 指令	程式語言一	開發環境建置
執行檔	executable	編譯後可直接執行的檔案，如 `hello`／`hello.exe`	程式語言一	開發環境建置
終端機	terminal	用文字指令操作電腦的視窗，在這裡下 `gcc`、`./hello`	程式語言一	開發環境建置
環境變數	PATH	系統尋找指令的路徑清單；GCC 沒加進去就會「找不到 gcc」	程式語言一	開發環境建置
版本控制	Git	記錄程式碼修改歷史、可回溯的工具	程式語言一	開發環境建置
遠端倉庫	GitHub	放 Git 專案的雲端平台，本課用來繳交作業	程式語言一	開發環境建置

§ 2 核心概念

核心概念

電腦只認得機器碼（0 與 1），但你用 C 寫的是人看得懂的文字。中間需要一個翻譯官，把 `hello.c` 翻成執行檔，這個翻譯官就是**編譯器（compiler）**。

所以開發環境要裝**兩個不同角色**的工具：**編輯器**（VS Code，用來「寫」）與**編譯器**（GCC，用來「翻譯」）。本課刻意把兩者分開、不用把它們包在一起的 IDE（如 Dev-C++），目的是讓你看清每個工具在做什麼，這個理解學任何語言都通用。

§ 3 主要內容

3.1 編譯在做什麼

`gcc hello.c -o hello` 一個指令，背後其實跑了四個階段。今天不需要全懂，只要知道「gcc 就是做這件事的工具」。

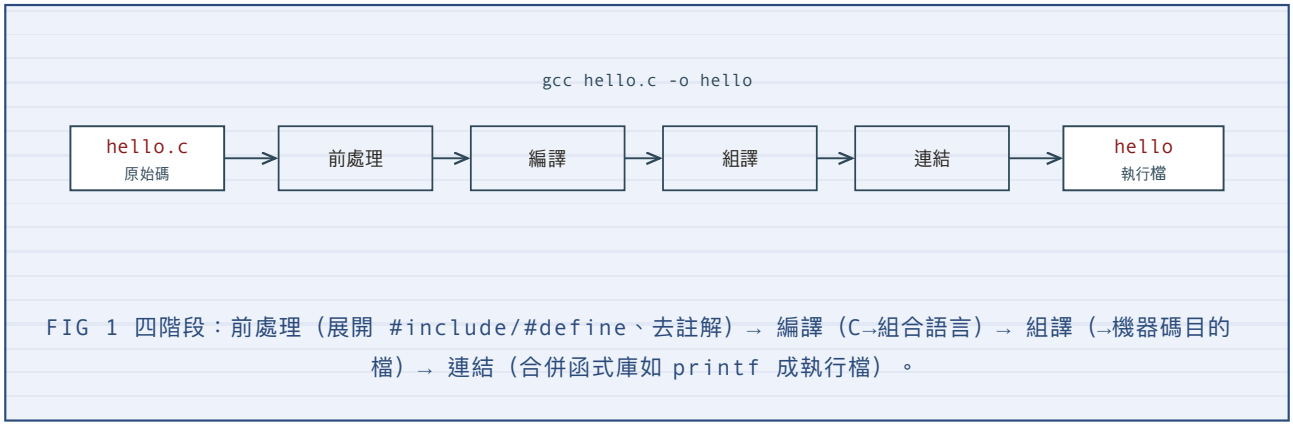

FIG 1 四階段：前處理（展開 #include/#define、去註解）→ 編譯（C→組合語言）→ 組譯（→機器碼目的檔）→ 連結（合併函式庫如 printf 成執行檔）。

3.2 安裝 VS Code（編輯器）

到 code.visualstudio.com 下載對應作業系統版本。安裝後：

1. Windows 安裝時勾選「Add to PATH」「Open with Code」。
2. 開 VS Code → 左側 Extensions → 搜尋並安裝 Microsoft 官方「C/C++」擴充套件。
3. （選用）安裝「Chinese (Traditional)」語言包。

一定要會的操作：用 `File → Open Folder`（永遠開資料夾、不要開單檔）；開終端機 `Ctrl + ``（反引號）；C 檔存成 .c 結尾；隨手 Ctrl + S`（檔名旁有 ● 代表未存）。

3.3 安裝 GCC（編譯器）

- **Windows**：裝 MSYS2（msys2.org）→ 在 MSYS2 終端機 `pacman -S mingw-w64-ucrt-x86_64-gcc` → 把 MSYS2 的 `bin` 加進系統 PATH。
- **macOS**：終端機輸入 `xcode-select --install`，會裝 Apple 的 Clang（相容 gcc 指令；之後輸入 `gcc` 實際呼叫的是 clang）。

驗證：`gcc --version` 看到版本資訊即成功；若顯示「'gcc' is not recognized / command not found」代表 PATH 沒設好。

3.4 第一次編譯與執行

建 `hello.c`：

```c
#include <stdio.h>
int main(void) {
    printf("Hello, World!\n");
    return 0;
}
```

終端機：

```
$ gcc hello.c -o hello
$ ./hello
Hello, World!
```

看到輸出就代表環境可用了。

3.5 專案資料夾

每門課一個資料夾、每週一個子夾；**檔名與路徑全用英文、不要中文或空格**（中文路徑可能害編譯器出錯）。

```
C-Programming/
├── week01/  (hello.c, test.c)
├── week02/
└── ...
```

3.6 Git 與 GitHub（作業繳交）

Git 是記錄修改歷史的工具，GitHub 是放專案的雲端平台。最少要會：`git init`（建倉庫）、`git add .`（加入變更）、`git commit -m "訊息"`（存一個版本）、`git push`（上傳到 GitHub）。細節由助教在實驗課帶。

§ 4 語法與函式速查

```
# 編譯與執行
gcc hello.c -o hello        # 編譯，輸出執行檔 hello
./hello                     # 執行（Windows: hello 或 hello.exe）

# 建議的標準編譯指令
gcc -Wall -std=c11 hello.c -o hello

# 終端機常用
cd 資料夾    # 切換目錄    ls(dir)  # 列出檔案

# Git 基本
git init / git add . / git commit -m "訊息" / git push
```

gcc 參數	作用
`-o 名稱`	指定輸出執行檔名稱
`-Wall`	顯示所有警告（強烈建議每次加）
`-std=c11`	使用 C11 標準，行為一致
`-g`	加除錯資訊（之後學除錯工具用）

§ 5 常見錯誤

常見錯誤

- `gcc` 找不到（not recognized / command not found）：PATH 沒設好，把編譯器路徑加進系統環境變數。
- 路徑或檔名用中文／空格：可能害編譯器出錯，一律用英文。
- 把 VS Code 當成 Visual Studio：兩者不同軟體，要下載的是 **Visual Studio Code**。
- 用「開啟檔案」而非「開啟資料夾」：VS Code 無法正確管理專案。
- 改了沒存檔（檔名旁 ●）就編譯：跑到的是舊版，養成 `Ctrl + S` 習慣。
- 忘了加 `-Wall`：警告被藏起來，潛在問題不會提醒你。

§ 6 練習題

練習 1（一般題）：裝好並驗證

裝好 VS Code（含 C/C++ 擴充）與 GCC 後，在終端機執行 `gcc --version`，把看到的版本資訊抄下來。

> **引導步驟**
>
> 1. 開 VS Code 終端機（`Ctrl + `` ）。
> 2. 輸入 `gcc --version`。
> 3. 看到版本＝成功；看到「not recognized / command not found」＝回去設 PATH。

解答

成功時類似 `gcc (GCC) 13.2.0 …`（macOS 會顯示 `Apple clang version …`，正常）。失敗多半是 PATH 沒設好。

練習 2（重要題）：編譯並執行 hello.c

建立 `hello.c` 印出 `Hello, World!`，用 `gcc -Wall -std=c11 hello.c -o hello` 編譯後執行，記錄指令與輸出。

> **引導步驟**
>
> 1. `File → Open Folder` 開一個英文路徑的資料夾，新增 `hello.c`。
> 2. 貼入 hello world 程式碼，`Ctrl + S` 存檔。
> 3. 終端機 `gcc -Wall -std=c11 hello.c -o hello`，再 `./hello`。

解答

```
$ gcc -Wall -std=c11 hello.c -o hello
$ ./hello
Hello, World!
```

`-Wall` 沒跳警告代表程式乾淨；`-o hello` 指定執行檔名；`./` 表示執行「目前資料夾裡」的這支。

易錯

- 忘了 `./` 直接打 `hello`：某些系統會找不到。
- 改了程式碼沒重新 `gcc` 就 `./hello`：跑到舊版。

§ 7 自我檢核

- [ ] 能用一句話說明編譯器在做什麼（原始碼 → 機器碼）。
- [ ] 分得清編輯器（VS Code）與編譯器（GCC）的角色。
- [ ] 裝好 VS Code 並安裝 C/C++ 擴充套件。
- [ ] 裝好 GCC，`gcc --version` 能看到版本。
- [ ] 能在終端機用 `gcc hello.c -o hello` 編譯、`./hello` 執行。
- [ ] 知道標準編譯指令 `gcc -Wall -std=c11` 各參數意思。
- [ ] 用英文路徑、每週一子資料夾的結構。
- [ ] 知道 Git/GitHub 是版本控制與作業繳交，會基本 init/add/commit/push。