變數與記憶體

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
變數	variable	一塊有名字的記憶體空間，用來存資料	程式語言一	變數與記憶體
宣告	declaration	告訴編譯器變數的型態與名字，並配置空間	程式語言一	變數與記憶體
初始化	initialization	宣告同時給初值，如 `int x = 0;`	程式語言一	變數與記憶體
資料型態	data type	決定變數佔幾 byte、能存什麼、怎麼解讀	程式語言一	變數與記憶體
取大小	sizeof	回傳型態或變數佔用的位元組數	程式語言一	變數與記憶體
整數溢位	integer overflow	整數超過型態範圍，繞回另一端	程式語言一	變數與記憶體
未定義行為	undefined behavior	規格未定義（如用未初始化變數），結果不可預測	程式語言一	變數與記憶體
浮點精度	floating-point precision	浮點以二進位近似，`0.1+0.2 ≠ 0.3`	程式語言一	變數與記憶體
ASCII	ASCII	字元與整數的對應表，`'A'` = 65	程式語言一	變數與記憶體
常數	const	宣告後不可修改的具型態變數	程式語言一	變數與記憶體
巨集	macro (#define)	前處理器做的純文字替換，無型態	程式語言一	變數與記憶體

§ 2 核心概念

核心概念

變數就是一塊有名字的記憶體空間。`int age = 20;` 一行做三件事：① 宣告型態 `int`（告訴編譯器配置 4 bytes）② 給名字 `age`（這塊空間的標籤，之後用名字存取）③ 初始化 `= 20`（把 20 寫進那塊空間）。

型態決定三件事：佔幾 byte、能存什麼範圍、那串 byte 怎麼被解讀。所以同樣的位元，用 `int` 和用 `float` 看會是不同的值。

§ 3 主要內容

3.1 變數在記憶體裡長怎樣

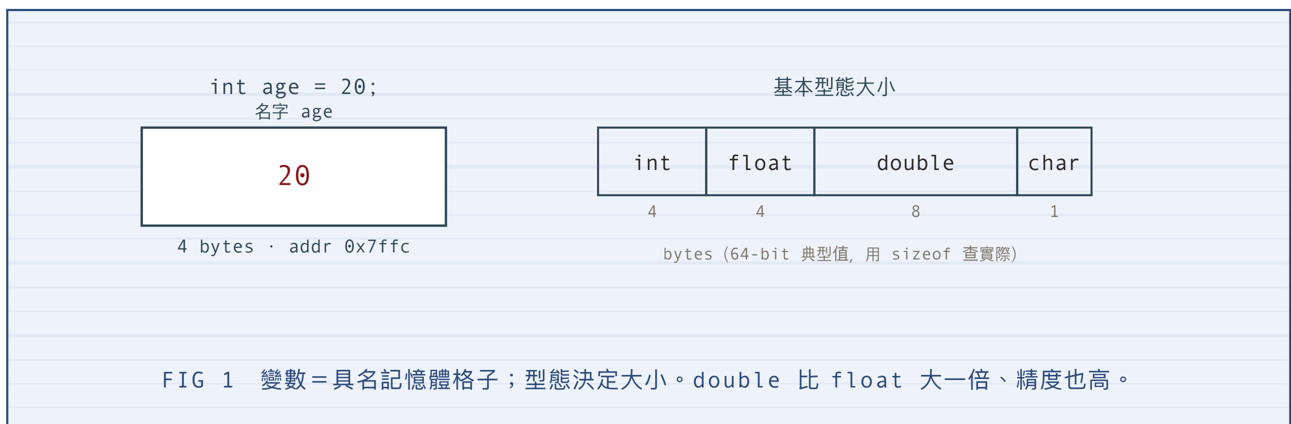

FIG 1 變數＝具名記憶體格子；型態決定大小。double 比 float 大一倍、精度也高。

3.2 命名規則

字母或底線開頭，只含字母／數字／底線，**區分大小寫**（`age ≠ Age`），不可用保留字（`int`、`float`⋯）。風格主流兩種：`snake_case`（C 社群，本課建議）與 `camelCase`。同一專案保持一致最重要。好命名讓人一看就懂（`student_count` 勝過 `n`）。

3.3 整數型態與溢位

`int` 4 bytes = 32 bits，1 bit 給正負號，範圍約 ±21 億（`2³¹-1 = 2147483647`）。**到最大值再 +1 會繞回最小負數**（`-2147483648`），這叫整數溢位，不會 crash 但值錯，是很多漏洞的根源。家族：`short`(2)、`int`(4)、`long`(4或8)、`long long`(8)；加 `unsigned` 去掉負號讓正數範圍加倍。初學用 `int` 就夠。

3.4 浮點：float vs double 與精度陷阱

`float` 4 bytes 約 7 位有效數字；`double` 8 bytes 約 15 位，**精度高、是預設選擇**。浮點以二進位近似，`0.1 + 0.2` 實際是 `0.30000000000000004`，所以 `0.1+0.2 != 0.3`（IEEE 754 通病，Python/Java 也一樣）。比較浮點別用 `==`，用容差：`fabs(a - 0.3) < 1e-6`。`float` 字面值要加後綴 `f`（`3.14f`）。

3.5 char 其實是整數

`char` 1 byte，存的是 ASCII 整數：`'A'` = 65、`'a'` = 97、`'0'` = 48。所以能做算術：`'A' + 1` = `'B'`、`'A' + 32` = `'a'`（大寫轉小寫）、`c - '0'` 把數字字元轉成整數。`%c` 印字元、`%d` 印它的 ASCII 值。**`'0'`（字元，48）和 `0`（整數）完全不同**。

3.6 宣告 vs 初始化

只宣告不初始化（`int x;`）→ `x` 是記憶體殘留的**垃圾值**，每次執行可能不同，用它是**未定義行為**，C 最常見 bug 來源之一。**永遠初始化你的變數**（`int x = 0;`）。

3.7 常數：const vs #define

`const double PI = 3.14159;` 有型態檢查、推薦；改它會編譯錯。`#define PI 3.14159` 是前處理器純文字替換、無型態。優先用 `const`。

§ 4 語法與函式速查

```
int age = 20;              // 宣告 + 初始化
int a = 1, b = 2;          // 一行多個（不建議）
const double PI = 3.14159; // 常數（有型態）
#define MAX 100            // 巨集（文字替換）

sizeof(int)     // 4（位元組數）
sizeof(x)       // 變數 x 佔的 byte

// 型態 ↔ 格式符
%d int    %f/%.2f float/double    %c char(字元)    %d char(ASCII值)
```

§ 5 常見錯誤

常見錯誤

- 用未初始化變數（垃圾值，UB）：宣告時就給初值。
- 整數溢位：超過 `int` 範圍會繞回負數，需要大數用 `long long`。
- 用 `==` 比較浮點：改用容差 `fabs(a-b) < 1e-6`。
- 把 `'0'` 當成 `0`：`'0'` 是字元（ASCII 48）。
- 變數名數字開頭或用保留字（`3d_score`、`int`）。
- `float` 字面值漏後綴 `f`；用 `int` 存有小數的值（被截斷）。

§ 6 練習題

練習 1（一般題）：char 算術

寫出輸出。

```
char c = 'C';
printf("%d\n", c - 'A');
printf("%c\n", 'a' - 32);
```

引導步驟

1. `'C'`=67、`'A'`=65。
2. `'a'`=97，減 32 是哪個字元的 ASCII？

解答

```
2
A
```

`67 - 65 = 2`；`97 - 32 = 65 = 'A'`（小寫轉大寫）。

練習 2（一般題）：型態與溢位

(a) 存「平均成績 87.5」該用什麼型態、為什麼？ (b) `int x = 2147483647; x = x + 1;` 之後 `x` 是多少？

解答

(a) `double`（或 float）：有小數，用 `int` 會截斷成 87。(b) `-2147483648`：到 `int` 最大值再 +1 整數溢位、繞回最小負數（不會 crash）。

練習 3（重要題）：抓 bug

下列程式為何每次跑可能印不同值？怎麼修？

```
#include <stdio.h>
int main(void) {
    int sum;
    for (int i = 1; i <= 5; i++) sum += i;
    printf("%d\n", sum);
    return 0;
}
```

引導步驟

1. `sum` 有沒有初值？第一次 `sum += i` 加到什麼上面？
2. 未初始化變數的值是什麼？

解答

`sum` 沒初始化，初值是垃圾值（未定義行為），`sum += i` 從垃圾值開始累加，結果不可預測。修：`int sum = 0;`。修好後輸出 `15`。

易錯

- 以為區域變數會自動歸零（不會；全域變數才會）。

§ 7 自我檢核

- ☐ 能說出 `int age = 20;` 做的三件事。
- ☐ 知道 int/float/double/char 的大小與用途，會用 `sizeof` 查。
- ☐ 知道整數溢位會繞回、不會 crash。
- ☐ 知道 `0.1+0.2 != 0.3`，浮點比較要用容差。
- ☐ 知道 char 是 ASCII 整數，會用 `'A'+1`、`c-'0'` 這類算術。
- ☐ 永遠初始化變數，知道未初始化是 UB。
- ☐ 分得清 `const` 與 `#define`。