問題：右邊的程式會停嗎？ 如果會，為什麼？

```
int main() {
    int i, j;
    for (i=0; i<100, j!=3; i++, j++) {
        printf("i=%d\n", i);
    }
    return 0;
}
```

答：一定會停止，但要花的時間（CPU TIME）不一定。其中緣由，要從三個面向分析，分別是：(1) 變數初始化；(2) 逗號 (comma operator)；(3) for 的 condition。

※文長，看**結論**請直接跳到最後（點我跳到最後）。

(1) 變數初始化

根據 C99 標準，靜態變數(static variable)、全域變數(global)等類型的變數在宣告時若未經初始化（賦值），則一律給予初始值，如 `int` 型別指定為 `0`，pointer 則設定為 null 指標，在這些情況下，不用考慮 garbage value 的問題。

然而在本題給的程式碼中，無論是 `i` 或是 `j`，因為在宣告時沒有賦值，都有 indeterminate value 的潛在問題。只不過，沒有定義好的行為，就是 compiler 自由發揮的時候，以 gcc 為例，在 `main()`之中的變數，如果沒有給予初始值，是有可能被 compiler 自動初始化的，那麼在 gcc 的編譯下，`i` 與 `j` 有可能自動以 `0` 為初始值。

(2) 逗號 (comma operator)

逗號（`,`）在 C 語言之中有兩種功用：

(a) operator，是二元運算子（binary operator）。

(b) separator，如用在 function call 與變數宣告。

本題出現的逗號屬於 operator，以下將專注在這個議題上。

逗號作為運算子（operator），其執行模式為由右至左執行指令，但只保留最右方指令的結果，如下：

```
int a = (1, 2, 3);
```

等同於 `int a = 3;`的結果。

(3) for 的 condition

for 迴圈內容被執行的前提是 condition 為 TRUE，其中所謂的 condition 指的是夾在兩個分號之中的區塊（即 `i<100, j!=3`），如下黃標背景所示：

```
for(i=0; i<100, j!=3; i++, j++) {
    printf("i=%d\n", i);
}
```

引述前面分析的第(2)點有關逗號運算子的功能，我們可以把這段 for 迴圈改成以下程式碼，就執行結果而言，沒有任何差異：

```
for(i=0; j!=3; i++, j++) {
    printf("i=%d\n", i);
}
```

因為 `i<100` 是不涉及任何更動變數內容的最單純之「比較運算」，其存在唯一價值是比較的結果，但這唯一的價值卻被逗號運算子無情捨棄掉，簡言之，即使把 `i<100` 直接去掉，也不會對執行結果造成任何影響；更明確地說，`i<100` 在這裡除了多消耗那麼一丁點兒 computer 的計算資源、以及可能造成 programmer 的思路混淆之外，完全沒有起到半毛作用。除非以邏輯運算子（如：&&、||等等）把多重條件結合在一起，又或者設計會更動變數內容的指令，原本在逗號左邊的指令才有存在的意義，如下範例，必須兩條件同時成立，結果才為 TRUE（這種作法等於摒棄逗號）：

```
for(i=0; (i<100) && (j!=3); i++, j++) {
    // to do something
}
```

或是下面這樣，只要有一個條件成立，迴圈便終止：

```
for(i=0; (i<100) || (j!=3); i++, j++) {
    // to do something
}
```

或者是下面這種情況，雖然 `i+=3` 的結果仍會被捨棄，但是 `i` 的內容已經變更，那麼 `i+=3` 才可能有其存在意義：

```
for(i=0; i+=3, j!=3; i++, j++) {
    // to do something
}
```

總結

我們可以發現，**本題的 for 迴圈，其終止的唯一條件是 j == 3**。所以問題便是 j 到底可不可能變成 3？**答案是一定會**，但可能有快或慢。

(1) **快的情況**：

變數 j 的初始值為 0，那麼只要三次迴圈執行就會終止，j 的變化為 0→1→2。但 j 的初始值一定為 0 嗎？這就要看情況，準確地說是看 compiler 以及所選用的參數。

(2) **慢的情況**：

如果 j 的初始值不是 0，那就要看是落在哪一個區塊，但一定最後會變成 3，因為題目有讓 j 不斷遞增（即 j++指令），故最糟的情況（worst case）是 j=4 開始，若以 32-bit signed int 來看，j 要歷經從 4 到 2^31-1=2147483647，然後 overflow 變成-2^31=-2147483648，再慢慢遞增到 3，也就是要執行

(2147483647-4) + 1 + (3+2147483648) = 4294967295 = 2^32-1

次 for 迴圈才會終止，但總歸還是要終止的。