由片語學習C程式設計

台灣大學資訊工程系劉邦鋒著

台灣大學劉邦鋒老師講授

August 19, 2016

第八單元

函式

- 函式是組成程式的基本元件。
- `main` 就是一個函式。
- 目前所介紹過的程式範例中。 整個程式只有一個 `main` 函式，本章節將開始介紹在一個程式中使用多個函式的方法。

函式分類

- **系統定義函式**
  系統定義函式是指，系統內已經有定義，使用者不需定義即可使用的函式。例如 `printf`。
- **使用者定義函式**
  使用者定義函式是指系統內沒有定義，使用者需要定義才能使用的函式。例如 `main`。

範例程式 1: (sys-function.c) 呼叫系統定義函式。

```
1  #include <stdio.h> /* for printf scanf */
2  #include <stdlib.h> /* for abs */
3  #include <math.h> /* for sin */
4  main()
5  {
6     int i, j;
7     double x, y;
8     scanf("%d", &i);
9     j = abs(i);
10    printf("%d\n", j);
11    scanf("%lf", &x);
12    y = sin(x);
13    printf("%f\n", y);
14    return 0;
15 }
```

輸入

```
1 -100
2 1.23
```

輸出

```
1 100
2 0.942489
```

標頭檔

- 使用系統定義函式必須引入想呼叫的函式對應的**標頭檔**。
  - 想呼叫 `abs`，則必須引入 `stdlib.h`。
  - 想呼叫 `printf` 及 `scanf`，則必須引入 `stdio.h`。
- 標頭檔中有詳細的函式的呼叫方法，這樣編譯器就能幫我們檢查呼叫方法是否正確。
  - `stdlib.h` 中有詳細的 `abs` 呼叫方法。
  - `stdio.h` 中有詳細的 `printf` 及 `scanf` 的呼叫方法。

```
1 #include <stdio.h> /* for printf scanf */
2 #include <stdlib.h> /* for abs */
3 #include <math.h> /* for sin */
```

- 最後面的副檔名 .h 是 header file 的意思。
- #include，是 引入 的意思。
- 用 < 及 > 將標頭檔包含起來。

函式原型

- 函式名稱，參數，及回傳值合稱為函式的**原型**。
- 函式的原型就好像是函式的使用說明書，詳細記載函式應該如何使用。
- 系統函式的原型都是定義在對應的標頭檔內。
- 編譯器看了標頭檔內的函式使用說明書之後， 就能判斷我們呼叫該函式的方式是否正確。

函式原型 2: (abs)

```
1 int abs(int n);
```

- 函數名稱為 abs。 函數名稱的命名原則和變數相同。
- 函數只有一個參數，類別為 int，參數必須用小刮號括起來，放在函數名稱後面。 呼叫時必須類別正確，除了變數也可以是算式。
- 函數有一個回傳值，類別為 int，必須宣告在函數名稱前面。
- 直接在原型後面加上分號表示結束。意即函式原型只會說明函式應該如何使用，而非函式內部如何實作。

實際呼叫

```
1   j = abs(i);
```

- 檢視之前呼叫 abs 函式的部分，並與 abs 函式的原型對照，
- 確認函式名稱、 參數、 及回傳值都沒有問題。
  - 函數名稱確實為 abs，沒有拼錯。
  - 函數確實只有一個參數 i，且類別為 int。確實用小括號括起來，放在函數名稱後面。
  - 接受函數回傳值的變數 j 確實為 int。

函式原型 3: (sin)

```
1 double sin(double x);
```

- 函數的名稱是 sin，計算正弦 (sine) 函數。
- 函數只有一個參數，類別為 double，以徑度表示。
- 函數有一個回傳值，類別為 double，為所求之正弦函數值。

實際呼叫

```
1   y = sin(x);
```

- 檢查呼叫 sin 的部分。 並與 sin 函式的原型對照，
- 函數的名稱確實為 sin，沒有拼錯
- 函數確實只有一個參數 x，且類別為 double。
- 接受函數回傳值的變數 y 確實為 double。

正確函式呼叫方法

- 使用正確的函數的名稱。
- 使用正確數量及資料類別的參數，參數必須以小括號括起來，而且如果有超過一個參數就要以逗號分開。
- 將函數回傳值做正確的處理。

系統函式回傳值

- 檢查系統函式回傳值是很重要的。
- 藉由檢查系統函式的回傳值我們可以知道系統函式是否已正確完成。
- `scanf` 的回傳值是有幾個變數已被正確讀入。如果已經沒有任何資料可供輸入，則 `scanf` 回傳 `EOF`，代表 end of file。

片語 4: 藉由 scanf 的回傳值判定是否還有資料.

```
1 while (scanf("%d", &data) != EOF) {
2   ...
3   process data;
4 }
```

範例程式 5: (scanf-count.c) 藉由 scanf 的回傳值掌握資料個數.

```
1  #include <stdio.h>
2
3  int main()
4  {
5     int sum = 0;
6     int count = 0;
7     int i;
8
9     while (scanf("%d", &i) != EOF) {
10       sum += i;
11       count++;
12    }
13    printf("%d\n", sum / count);
14    return 0;
15 }
```

輸入

```
1 98
2 89
3 87
4 99
5 96
```

輸出

```
1 93
```

學習要點

我們必須從鍵盤輸入特定的符號，讓程式知道已經沒有輸入了。在微軟的視窗系統我們可以鍵入 `ctrl-Z`， 在 *UNIX* 系統我們可以鍵入 `ctrl-D`。

學習要點

如果函式有回傳值但沒有回傳給任何人，這樣是可接受的。 重點是接受函式回傳值的變數類別與函數原型所描述的是否一致。

程式庫

- 程式庫 (library) 是將預先寫好並編譯好的常用的函式集合在一起，讓使用者程式可以直接呼叫使用的一個機制。
- 使用者寫好的程式經過編譯之後，就需要和用到的程式庫連結 (link)。所謂連結就是將各個函式之間的呼叫關係整理清楚，確定所有用到的函式都有定義，以便程式正確執行。
- 一般來說，使用者無須做特別的事情， 編譯器就會將所有 `printf`、 `scanf`、 `abs` 等 **C 程式語言標準程式庫**中的函式的連結整理清楚。

數學程式庫

- 有的函式，如 `math.h` 的 `sin`，不在 C 程式語言 的標準程式庫中，而是在另一個**數學程式庫** 之中。
- 如果編譯器不知道要與數學程式庫連結，則編譯器就無法幫我們建立完整的執行檔， 因為編譯器不知道如何產生計算 `sin` 的程式碼。因為就算引入 `math.h`，那也只是如何呼叫 `sin` 的說明書。真正計算 `sin` 的程式碼是在數學程式庫之中。

```
gcc sys-function.c -o system.exe -lm
```

- gcc 編譯器有一個連結程式庫的選項 -l。-l 後面接要連結的程式庫。
- 數學程式庫的全名是 libm.a，可以在 gcc 安裝目錄的 lib 下找到。
- 如果我們下 -lm 的選項，gcc 就會去找 libm.a，於是就能找到 sin 的程式碼，編譯器就能產生執行檔。

片語 6: 自己定義一個回傳一個整數的函式。

```
1 int myfunction(int i)
2 {
3   int value;
4   ...
5   compute value according to i;
6   ...
7   return value;
8 }
```

- 定義一個接受一個整數參數 i，並回傳一個整數的函式 myfucntion。
- 先寫回傳值的類別 int，再寫函式的名稱 myfunction，最後用小括號 ( ) 將參數的類別 int，及名稱 i 括起來。
- 此時不只描述函式的原型，也要定義函式如何實作，所以不能像函式原型直接用分號結束，而是要用一對 { } 將實作的部份包起來，就像寫 main 主程式一樣。
- 因為我們要回傳一個整數，所以我們宣告一個整數變數 value，並根據參數 i 值計算 value，最後使用 return 命令將 value 回傳。

學習要點

`return` 命令會立刻回到之前呼叫此函式的程式部分，

片語 7: 回傳一個整數的 main

```
1 #include <stdio.h>
2 int main(void)
3 {
4   return 0;
5 }
```

- main 主程式是一個沒有參數，但有一個整數回傳值的函式。而且通常我們將回傳值設為 0。
- 由於 main 不需要任何參數，所以 main 後面的 ( ) 中以 void 來表示。
- void 這個字的意思就是 **沒有**。

範例程式 8: (leap-year-function.c) 定義一個函式決定閏年

```
1  #include <stdio.h>
2  int leap_year(int y)
3  {
4    int is_leap;
5    is_leap = (y % 400 == 0) ||
6      ((y % 4 == 0) && !(y % 100 == 0));
7    return is_leap;
8  }
9  int main(void)
10 {
11   int year;
12   int k;
13   scanf("%d", &year);
14   k = leap_year(year);
15   printf("%d\n", k);
16   return 0;
17 }
```

leap_year

- 定義一個 leap_year 函式決定閏年。
- leap_year 函式接受一個整數參數 y， 若 y 是閏年， 則回傳 1，否則 回傳 0。

輸入

```
1 2011
```

輸出

```
1 0
```

計算的分工

- 站在 main 的觀點，決定 year 是否為閏年的計算過程被隱藏起來了。
- 如何決定 year 是否為閏年的計算過程已經不重要了，因為 leap_year 會去計算，不需要 main 主程式去煩心。
- main 主程式只需要將 year 讀進來，交給 leap_year 計算結果，最後再印出答案即可。
- leap_year 只需要計算答案，不須處理輸出入。

函式的優點

- 主程式與函式分工合作，各司其職，寫程式的人就可以專心把自己的部分寫清楚，
- 可以重複使用寫好的函式。重複程式碼不僅會讓程式冗長難以理解，而且很容易在重複撰寫時出錯。如果使用已經過驗證的函式，則可避免這些麻煩。

範例程式 9: (leap-year-order.c) 交換 leap_year 及 main 的次序

```
1  #include <stdio.h>
2  int main(void)
3  {
4    int year;
5    int k;
6    scanf("%d", &year);
7    k = leap_year(year);
8    printf("%d\n", k);
9    return 0;
10 }
11 int leap_year(int y)
12 {
13   int is_leap;
14   is_leap = (y % 400 == 0) ||
15     ((y % 4 == 0) && !(y % 100 == 0));
16   return is_leap;
17 }
```

警告訊息

```
C:\SVN\book\C>gcc -Wall leap-year-order.c
leap-year-order.c: In function ‘main’:
leap-year-order.c:7: warning: implicit declaration
of function ‘leap_year’
```

- 編譯器 (如 gcc 並使用 -Wall) 在編譯範例時發出的警告。

- 編譯器只會由上到下一次掃過這個程式。
- 當編譯器看到 `k = leap_year(year);` 這句時，它並不知道 `leap_year(year)` 函式的原型，所以只能假定你的參數用法是對的，然後發出警告提醒你。
- `leap_year` 未經正式宣告回傳值及參數就被使用了，所以編譯器只好幫 `leap_year` 補足這些未明確設定的部分。

前置宣告

- 可以用前置宣告 (forward declaration) 的方式避免編譯器發出警告。
- 前置宣告和之後 `leap_year` 宣告完全相同，只是宣告完 `leap_year` 之後馬上用分號結束。
- 編譯器由上到下一次掃過這個程式時，會先看到 `leap_year` 的原型，自然就會知道 `leap_year` 的回傳值及參數。
- 等看到呼叫 `leap_year` 部分時，就知道 `leap_year` 的用法正確，所以不會發出警告。

範例程式 10: (leap-year-forward.c) 使用前置宣告

```
1  #include <stdio.h>
2  int leap_year(int y);
3  int main(void)
4  {
5    int year;
6    int k;
7    scanf("%d", &year);
8    k = leap_year(year);
9    printf("%d\n", k);
10   return 0;
11 }
12 int leap_year(int y)
13 {
14   int is_leap;
15   is_leap = (y % 400 == 0) ||
16     ((y % 4 == 0) && !(y % 100 == 0));
17   return is_leap;
18 }
```

- 第 2 行稱為函式 `leap_year` 的**宣告部分**，也就是**原型**。宣告部分 (原型) 將一個函數的回傳值及參數講清楚，
- 第 12 到 18 行稱為 `leap_year` 函式的**定義部分**。定義部分將一個函數要做什麼講清楚。
- 為了讓 `main` 能明白 `leap_year` 的使用法，我們將宣告部分與定義部分分開，並將宣告部分往前放，成為 **前置宣告**。

輸入

```
1 2011
```

輸出

```
1 0
```

範例程式 11: (leap-year-repeat.c) 重複輸入年分並計算閏年。

```
1  #include <stdio.h>
2  int leap_year(int y);
3
4  int main(void)
5  {
6    int year;
7    int k;
8    while (scanf("%d", &year) != EOF) {
9      k = leap_year(year);
10     printf("%d\n", k);
11   }
12   return 0;
13 }
14
15 int leap_year(int y)
16 {
17   int is_leap;
18   is_leap = (y % 400 == 0) ||
19     ((y % 4 == 0) && !(y % 100 == 0));
20   return is_leap;
21 }
```

輸入

```
1 1973
2 1984
3 1900
4 1964
5 2000
```

輸出

```
1 0
2 1
3 0
4 1
5 1
```

片語 12: 如何自己定義一個無回傳值的函式

```
1 void foo(int i)
2 {
3   ...
4   process according to i;
5   ...
6   return;
7 }
```

- 定義一個接受一個整數參數 `i`，但不回傳任何值的函式 `foo`。
- 使用 `void` 表示 `foo` 並沒有任何回傳值。`void` 不可省略，否則編譯器會假設回傳值類別為 `int`。

範例程式 13: (print-digits.c) 印出一個數的各位數

```
2  void print_digits(int i)
3  {
4    int index = 0;
5    int digits[20];
6    if (i < 0)
7      return;
8    while (i != 0) {
9      digits[index] = (i % 10);
10     i /= 10;
11     index++;
12   }
13   for (i = index - 1; i >= 0; i--)
14     printf("%d\n", digits[i]);
15   return;
16 }
```

```
18 int main(void)
19 {
20   int i;
21   scanf("%d", &i);
22   print_digits(i);
23   return 0;
24 }
```

- 將參數 `i` 的各位數分別印在不同行。
- 用一個 `while` 迴圈依序取出 `i` 的最後一位數放入 `digits` 陣列中，並用 `index` 記住下一次要放的位址。
- 將 `i` 除以 10，處理下一位數，而 `while` 會重複直到 `i` 變成 0。
- 最後倒著把 `digits` 陣列中的數字印出即可。

- `print_digits` 只是在做一些列印的工作，不需要回傳值，`return` 的後面就直接加分號結束。 (第 15 行)
- 使用的方法無法處理 `i` 小於 0 的狀況，所以使用 `return;` 立即跳回到 `main` 主程式。 (第 7 行)
- 與第 7 行的 `return` 不同，第 15 行的 `return`是可以省略的，但是我們還是放上 `return`，讓程式碼的讀者能夠清楚知道程式的流程。

輸入

```
1 467326
```

輸出

```
1 4
2 6
3 7
4 3
5 2
6 6
```

風格要點

在無回傳值函式的最後加上 `return` 可增加程式碼的可讀性。

範例程式 14: (year-month.c) 決定一個月有幾天

```
2  int leap_year(int y)
3  {
4    int is_leap;
5    is_leap = (y % 400 == 0) ||
6      ((y % 4 == 0) && !(y % 100 == 0));
7    return is_leap;
8  }
```

```
10 int how_many_days(int year, int month)
11 {
12   int days;
13   if (year < 0 || month < 1 || month > 12)
14     return 0;
15   switch (month) {
16     case 1: case 3: case 5: case 7:
17     case 8: case 10: case 12:
18       days = 31; break;
19     case 4: case 6: case 9: case 11:
20       days = 30; break;
21     case 2:
22       days = leap_year(year)? 29 : 28;
23       break;
24     default:
25       days = 0;
26     }
27   return days;
28 }
```

```
30 int main(void)
31 {
32   int year;
33   int month;
34   int days;
35   scanf("%d", &year);
36   scanf("%d", &month);
37   days = how_many_days(year, month);
38   printf("%d\n", days);
39   return 0;
40 }
```

用函式的觀念化簡程式

- 主程式中讀入 year 及 month 後呼叫 how_many_days 決定該月中有幾天。
- how_many_days 呼叫 leap_year 決定 year 是否閏年，才能決定天數。

輸入

```
1 2011 9
```

輸出

```
1 30
```

範例程式 15: (print-start-end.c) 印出 start 到 end

```
1  #include <stdio.h>
2  void print_numbers(int start, int end)
3  {
4    int i;
5    for (i = start; i <= end; i++)
6      printf("%d\n", i);
7    return;
8  }
9  int main(void)
10 {
11   int a, b;
12   scanf("%d", &a);
13   scanf("%d", &b);
14   print_numbers(a, b);
15   return 0;
16 }
```

輸入

```
1  3  9
```

輸出

```
1  3
2  4
3  5
4  6
5  7
6  8
7  9
```

printf, scanf

函式原型 16: (printf-scanf)

```
1 int printf(char *format, ... );
2 int scanf(char *format, ... );
```

- 第二個參數非常奇特，是 ...，意思是參數個數是不固定的。
- 之前 printf 及 scanf 時都一次處理一個變數，但是 "..." 不固定參數個數能讓我們同時對多個變數作輸出入。

片語 17: 對多個變數作輸出輸入

```
1 printf("%d %p %f %f\n", int, addr, float, double);
2 scanf("%d%f%lf", &int, &float, &double);
```

- 在第一個參數中有許多的 %d 或 %f 等項目，每一個 % 項目都必須與後面的參數一一對應。
- 因為不知道有多少個，於是只好使用 ...，表示參數個數是**不固定**的。
- 為了分辨印出的值是屬於哪一個變數，我們使用空白字元將 printf 第一個參數中的各個 %d 或 %f 隔開。

範例程式 18: (multi-io.c) 對多個變數作輸出輸入

```
1  #include <stdio.h>
2  int main(void)
3  {
4     int i;
5     float f;
6     double df;
7     scanf("%d%f%lf", &i, &f, &df);
8     printf("%d %p %f %f\n", i, &i, f, df);
9     return 0;
10 }
```

輸入

```
1 -1 3.2 4.6
```

輸出

```
1 -1 0x7fff95418260 3.200000 4.600000
```

片語 19: 對多個變數作輸出並夾雜其他字元

```
1 printf("%d%p%f%f\n", int, addr, float, double);
```

- 為了使 printf 所印出的訊息更有可讀性，我們可以將其他字元加入 printf 的第一個參數中，printf 就會將這些字元按照在第一個參數中的位置依序印出，

範例程式 20: (multi-io-message.c) 輸出夾雜其他字元

```
1  #include <stdio.h>
2  int main(void)
3  {
4     int i;
5     float f;
6     double df;
7     scanf("%d%f%lf", &i, &f, &df);
8     printf("int %d adr %p flt %f dbl %f\n",
9             i, &i, f, df);
10    return 0;
11 }
```

輸入

```
1 -1 3.2 4.6
```

輸出

```
1 int -1 adr 0x7fff98a1f330 flt 3.200000 dbl 4.600000
```

範例程式 21: (print-matrix-address-message.c) 印出方便閱讀的訊息

```
4   int a[2][3][4];
5   int i, j, k;
6   printf("sizeof(a[0][0][0]) is %d\n",
7          sizeof(a[0][0][0]));
8   printf("sizeof(a[0][0]) is %d\n",
9          sizeof(a[0][0]));
10  printf("sizeof(a[0])is %d\n",
11         sizeof(a[0]));
12  printf("sizeof(a) is %d\n",
13         sizeof(a));
14  for (i = 0; i < 2; i++)
15    for (j = 0; j < 3; j++)
16      for (k = 0; k < 4; k++)
17        printf("a[%d][%d][%d] at %p\n",
18               i, j, k, &(a[i][j][k]));
```

```
20    for (i = 0; i < 2; i++)
21      printf("address of a[%d][0] %p\n",
22              i, &(a[i][0]));
23    for (i = 0; i < 2; i++)
24      printf("value of a[%d][0] is %p\n",
25              i, a[i][0]);
26    for (i = 0; i < 2; i++)
27      printf("address of a[%d] is %p\n",
28              i, &(a[i]));
29    for (i = 0; i < 2; i++)
30      printf("value of a[%d] is %p\n",
31              i, a[i]);
32    printf("address of a is %p\n", &a);
33    printf("value of a is %p\n", a);
```

輸出

```
1  sizeof(a[0][0][0]) is 4
2  sizeof(a[0][0]) is 16
3  sizeof(a[0])is 48
4  sizeof(a) is 96
5  a[0][0][0] at 0x7fffa8124340
6  a[0][0][1] at 0x7fffa8124344
7  a[0][0][2] at 0x7fffa8124348
8  a[0][0][3] at 0x7fffa812434c
9  a[0][1][0] at 0x7fffa8124350
10 a[0][1][1] at 0x7fffa8124354
11 a[0][1][2] at 0x7fffa8124358
12 a[0][1][3] at 0x7fffa812435c
13 a[0][2][0] at 0x7fffa8124360
14 a[0][2][1] at 0x7fffa8124364
15 a[0][2][2] at 0x7fffa8124368
16 a[0][2][3] at 0x7fffa812436c
17 a[1][0][0] at 0x7fffa8124370
18 a[1][0][1] at 0x7fffa8124374
19 a[1][0][2] at 0x7fffa8124378
```

```
20 a[1][0][3] at 0x7fffa812437c
21 a[1][1][0] at 0x7fffa8124380
22 a[1][1][1] at 0x7fffa8124384
23 a[1][1][2] at 0x7fffa8124388
24 a[1][1][3] at 0x7fffa812438c
25 a[1][2][0] at 0x7fffa8124390
26 a[1][2][1] at 0x7fffa8124394
27 a[1][2][2] at 0x7fffa8124398
28 a[1][2][3] at 0x7fffa812439c
29 address of a[0][0] 0x7fffa8124340
30 address of a[1][0] 0x7fffa8124370
31 value of a[0][0] is 0x7fffa8124340
32 value of a[1][0] is 0x7fffa8124370
33 address of a[0] is 0x7fffa8124340
34 address of a[1] is 0x7fffa8124370
35 value of a[0] is 0x7fffa8124340
36 value of a[1] is 0x7fffa8124370
37 address of a is 0x7fffa8124340
38 value of a is 0x7fffa8124340
```

- scanf 的格式字串沒有放空格，print 的格式字串有放空格。
- 如果 print 的格式字串沒放空格，顯示的結果將會是連在一起，根本無法閱讀。
- 在鍵盤輸入時，我們**必須**使用空格，換行，或是 tab 字元將輸入的資料隔開。
- 就算 scanf 格式字串是 "%d%f%lf"，scanf 也能在輸入資料中找出一個整數，浮點數，和倍準浮點數。中間有換行，空格，tab 都沒關係。因為 scanf 會跳過換行，空白，tab， 在輸入中想辦法找到**下一筆資料**。

範例程式 22: (scanf-nonspace.c) scanf 必須在輸入資料裡找到相同的字元

```
1  #include <stdio.h>
2
3  int main()
4  {
5    int i = 1,  j = 2, k = 3;
6    int n;
7
8    n = scanf("%d/%d/%d", &i, &j, &k);
9    printf("i = %d,  j = %d, k = %d\n", i, j, k);
10   printf("%d items read by scanf\n", n);
11
12   n = scanf("%d/%d/%d", &i, &j, &k);
13   printf("i = %d,  j = %d, k = %d\n", i, j, k);
14   printf("%d items read by scanf\n", n);
15
16   return 0;
17 }
```

- 格式字串放入空白、 換行、 tab 並無任何作用，因為 scanf 會對它們置之不理。
- 如果 scanf 格式字串有**非空白**字元，scanf 就必須在輸入資料裡找到相同的字元，否則就回傳失敗。
- 第一次輸入的時候，因為輸入是 4/5/2011，確實是 %d/%d/%d 的格式，所以沒有問題。
- 第二次輸入的時候，是 6 /7/2012，不是 %d/%d/%d 的格式。由於格式字串在 6 後面是 /，而空白不是 /，剩下的輸入就被忽略了。

輸入

```
1 4/5/2011
2 6 /7/2012
```

輸出

```
1 i = 4,  j = 5, k = 2011
2 3 items read by scanf
3 i = 6,  j = 5, k = 2011
4 1 items read by scanf
```

學習要點

如果 scanf 格式字串有非空白字元，scanf 就必須在輸入資料裡找到相同的字元。

形式與實際參數

- 形式參數 (formal parameter) 就是寫在被呼叫方函式的宣告部分， 所以一定是一個變數的類別。例如 `j` 就是 `test` 的形式參數。
  - `void test(int j)`
- 實際參數 (actual parameter) 是呼叫方實際用以呼叫被呼叫函式的參數。實際參數可以是一個算式，並不一定是一個變數。
  - `test(i)`
  - `test(3 + 7)`

範例程式 23: (function-parameter.c) 印出參數的位址及值

```
1  #include <stdio.h>
2  void test(int j)
3  {
4    printf("test: the address of j = %p\n", &j);
5    printf("test: before adding 1 j = %d\n", j);
6    j++;
7    printf("test: after adding 1 j = %d\n", j);
8    return;
9  }
10 int main(void)
11 {
12   int i;
13   scanf("%d", &i);
14   printf("main: the address of i = %p\n", &i);
15   printf("main: before calling test i = %d\n", i);
16   test(i);
17   printf("main: after calling test i = %d\n", i);
18   test(3 + 7);
19   return 0;
20 }
```

形式與實際參數

- 在C 程式語言 中參數的傳遞是先將實際參數的值算出，再將這個值由呼叫方指定給被呼叫方，當作相對應形式參數的初始值。
- `main` 以變數 `i` 呼叫 `test` 時，會用 `i` 的值當成形式參數 `j` 的初始值。
- 以算式 3 + 7 呼叫 `test` 時， 會用 3 + 7 的結果，也就是 10 當成形式參數 `j` 的初始值。

輸入

```
1 5
```

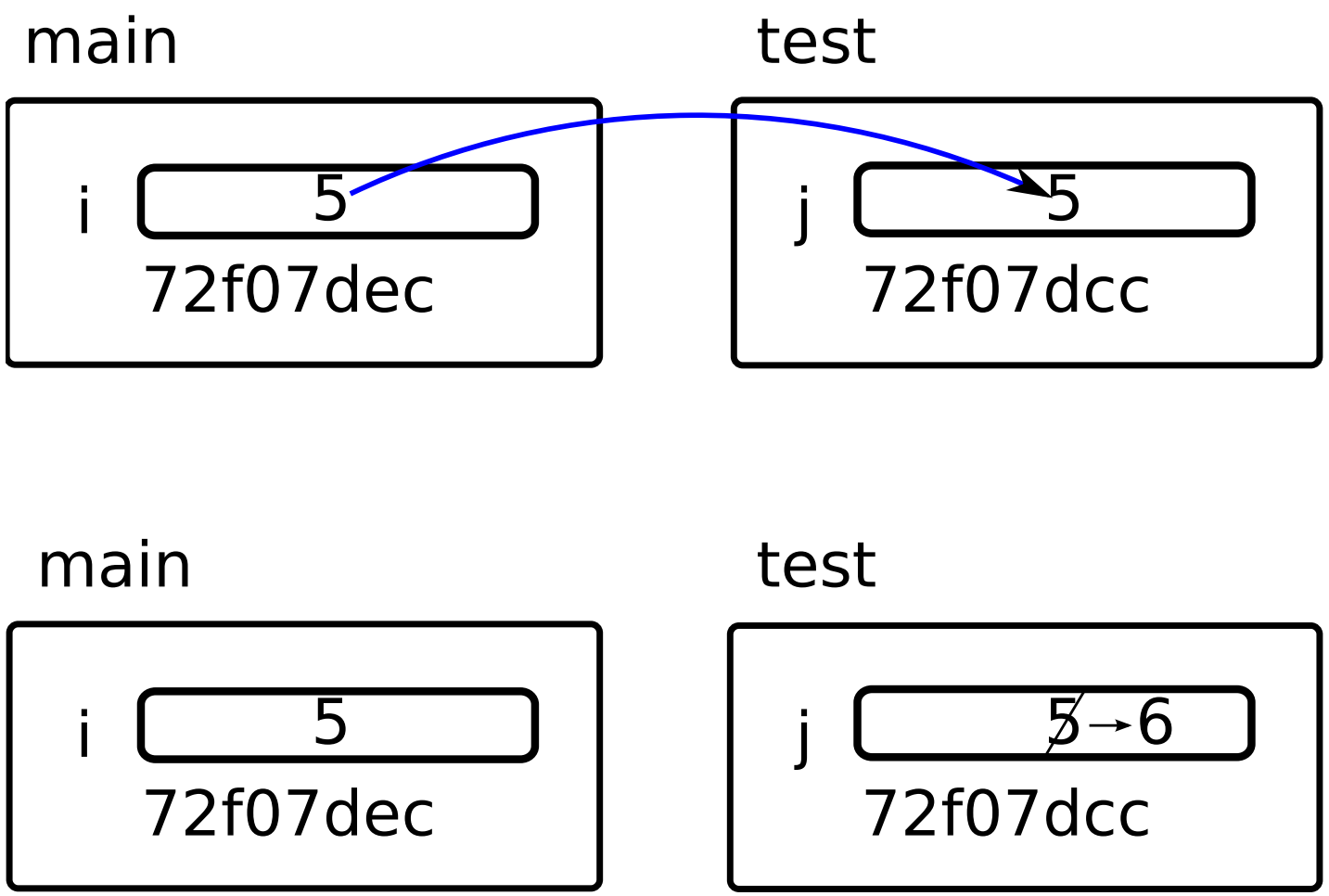
輸出

```
1 main: the address of i = 0x7fff72f07dec
2 main: before calling test i = 5
3 test: the address of j = 0x7fff72f07dcc
4 test: before adding 1 j = 5
5 test: after adding 1 j = 6
6 main: after calling test i = 5
7 test: the address of j = 0x7fff72f07dcc
8 test: before adding 1 j = 10
9 test: after adding 1 j = 11
```

- 首先主程式 main 先印出 i 的值作為之後比較用，再用變數 i 當作實際參數呼叫函式 test。
- 函式 test 中形式參數 j 就會得到由 main 傳來的實際參數 i 值。所以 j 的初始值即為 i 的值 5。
- 在 test 中我們將 j 加 1 成為 6 並印出。我們接著再回到 main，並將 i 再印出一次做比較。
- 最後我們再用 3 + 7 當作實際參數呼叫函式 test。函式 test 中形式參數 j 就會得到由 main 傳來的 10。

- main 中的實際參數 i 和 test 中的形式參數 j 是不同的變數，這可由執行結果的第 1 行及第 3 行顯示他們的記憶體位址不同得到證明。
- 既然 main 中的 i 和 test 中的 j 是不同的變數，我們在 test 中將 j 加 1 並不會影響 main 中的 i 的值。這可由執行結果的第 2 行及第 6 行得到證明。
- 實際參數可以是一個算式，所以我們可以使用算式呼叫函式。

呼叫函式


main
i
5
72f07dec
test
j
5
72f07dcc
main
i
5
72f07dec
test
j
5→6
72f07dcc

陣列參數傳遞

片語 24: 使用函數處理一個陣列中的元素

```
1  void process_array(int array[], int n)
2  {
3    int i;
4    for (i = 0; i < n; i++)
5      process element array[i];
6    return;
7  }
8  int main(void)
9  {
10   int a[10];
11   process_array(a, 10);
12   return 0;
13 }
```

- 宣告形式參數陣列 array 時寫成 array[]。
- C 程式語言在傳參數時，是把實際參數的值算出，交給形式參數。
- 當實際參數是一個陣列 a 時，a 的值就是陣列 a 的起始記憶體位址。形式參數陣列 array 所接到的的值就是一個 10 個元素的陣列 a 的起始記憶體位址。
- main 也可以用一個 20 個元素的陣列 b 當作實際參數來呼叫 process_array。此時形式參數陣列 array 所接到的的值就是一個 20 個元素的陣列 b 的起始記憶體位址。
- 既然形式參數陣列 array 將來所接到的的值可能是包含任意個元素的陣列，我們也無從宣告 array 的長度，所以就省略成 array[]，

範例程式 25: (print-array-with-function.c) 用函式印陣列

```
2 void print_array(int array[], int n)
3 {
4   int i;
5   printf("array is at %p\n", array);
6   for (i = 0; i < n; i++)
7     printf("array[%d] = %d\n", i, array[i]);
8   return;
9 }
```

```
11 int main(void)
12 {
13   int i;
14   int a[3];
15   int b[5];
16   printf("main: a = %p\n", a);
17   printf("main: b = %p\n", b);
18   for (i = 0; i < 3; i++)
19     scanf("%d", &(a[i]));
20   for (i = 0; i < 5; i++)
21     scanf("%d", &(b[i]));
22   print_array(a, 3);
23   print_array(b, 5);
24   return 0;
25 }
```

輸入

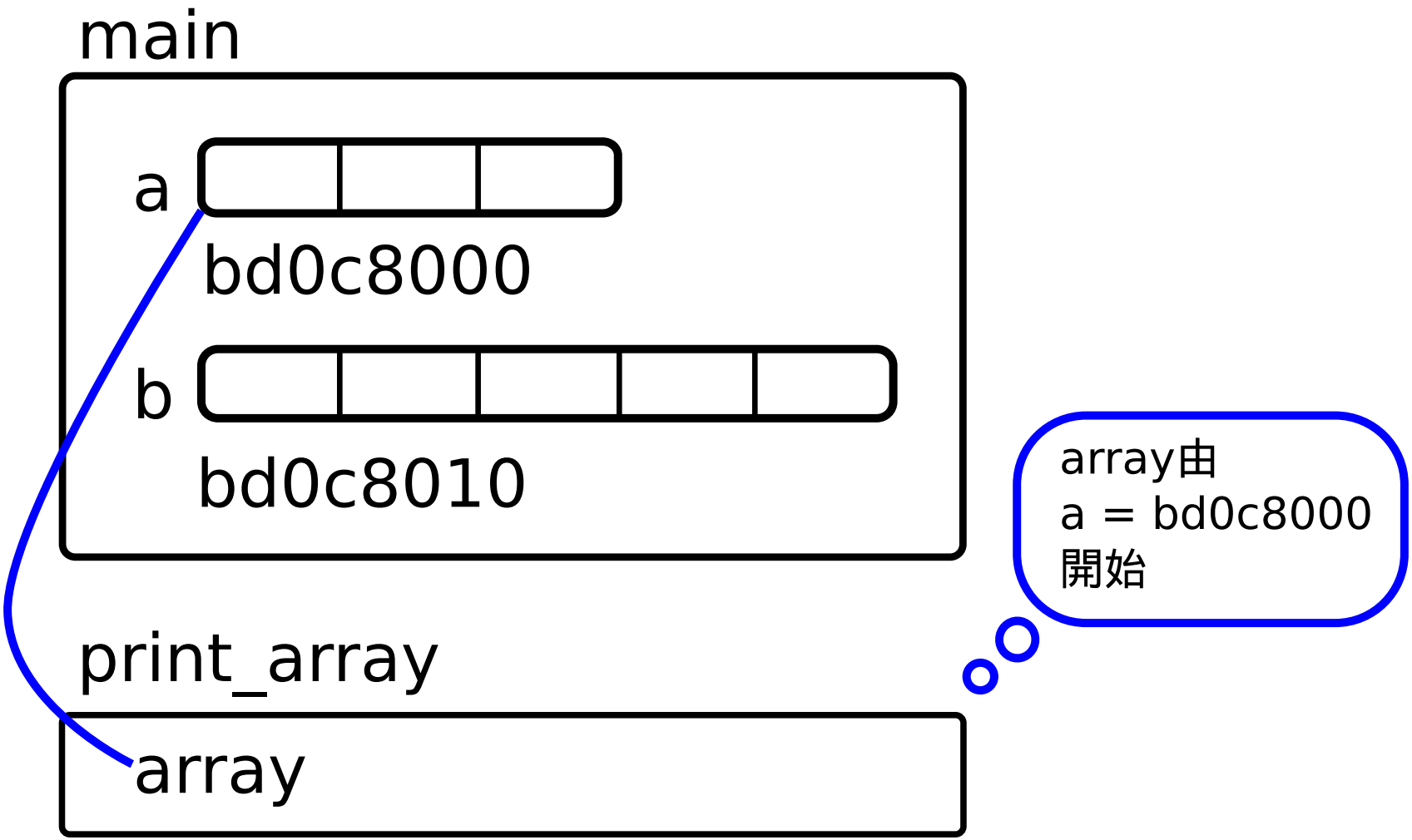
```
1  3 7 5
2  2 6 8 3 9
```

輸出

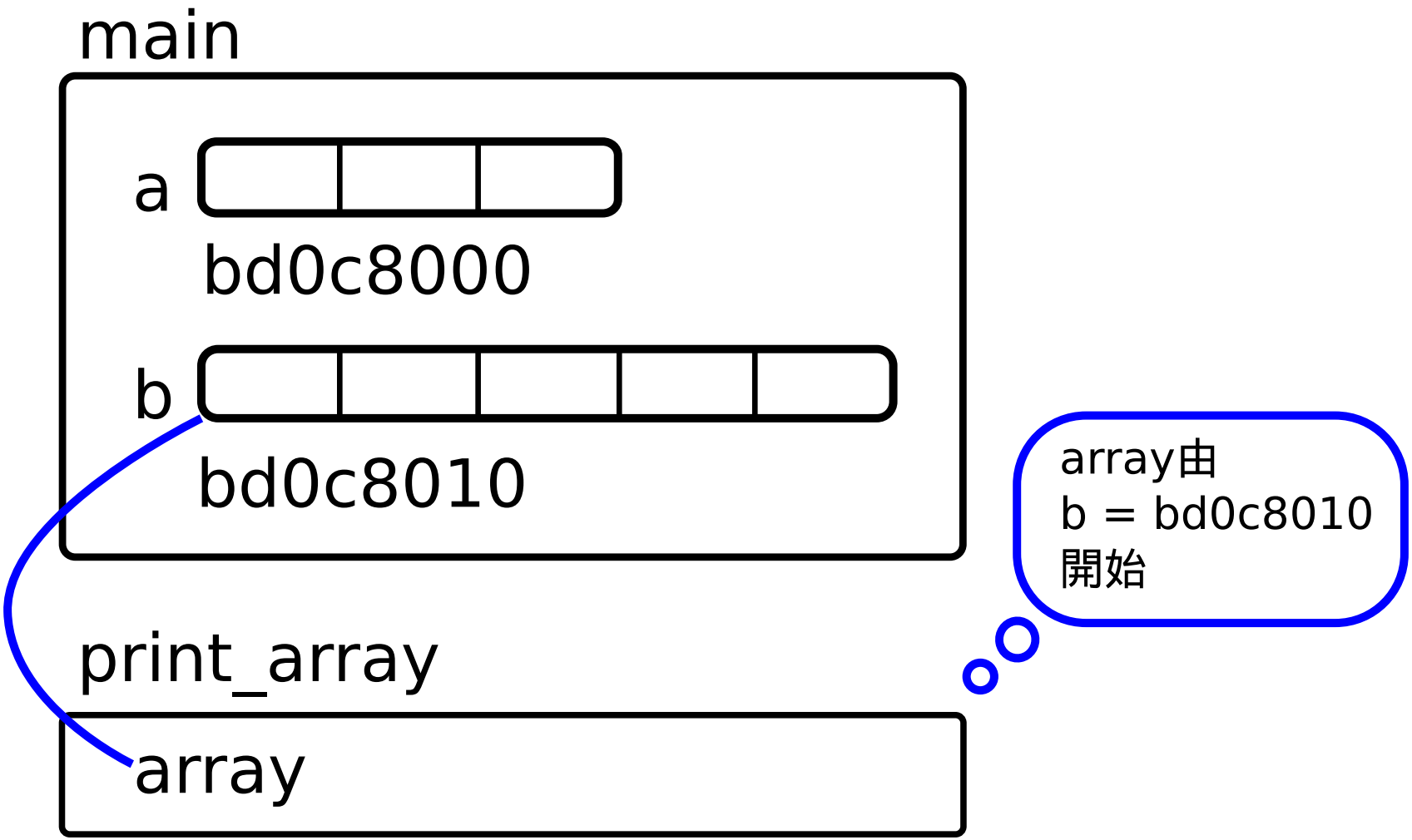
```
1  main: a = 0x7fffbd0c8000
2  main: b = 0x7fffbd0c8010
3  array is at 0x7fffbd0c8000
4  array[0] = 3
5  array[1] = 7
6  array[2] = 5
7  array is at 0x7fffbd0c8010
8  array[0] = 2
9  array[1] = 6
10 array[2] = 8
11 array[3] = 3
12 array[4] = 9
```

- 實際參數是陣列 a 時，形式參數陣列 array 的值就是陣列 a 的值，此時陣列 array[i] 就是陣列 a[i]。
- 實際參數是陣列 b 時，形式參數陣列 array 的值就是陣列 b 的值，此時陣列 array[i] 就是陣列 b[i]。

傳遞陣列


main
a
bd0c8000
b
bd0c8010
array由
a = bd0c8000
開始
print_array
array

傳遞陣列


main
a
bd0c8000
b
bd0c8010
array由
b = bd0c8010
開始
print_array
array

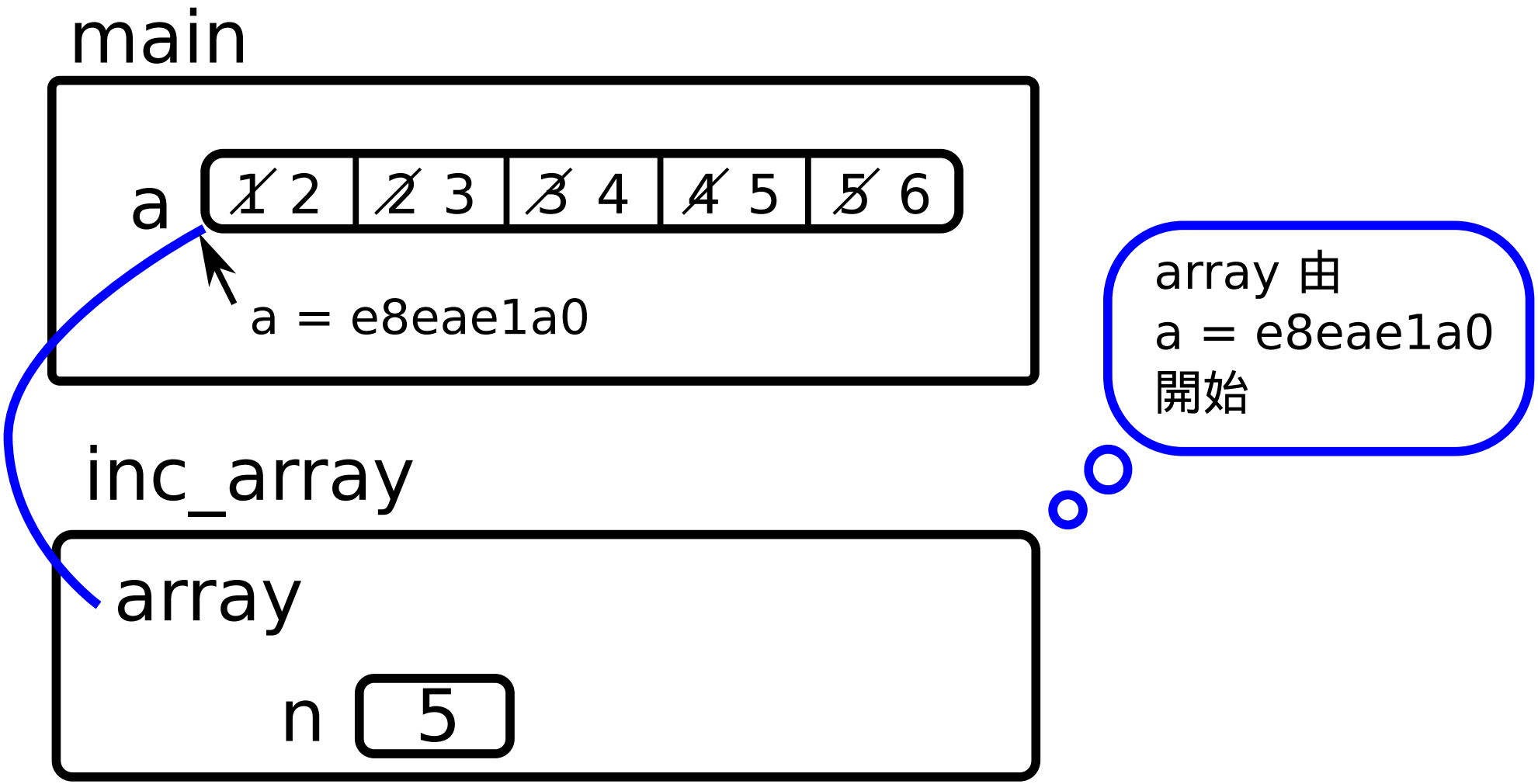
範例程式 26: (partial-inc.c) 增加部分陣列元素的值

```
 9  void inc_array(int array[], int n)
10  {
11    int i;
12    printf("inc_array: array = %p\n", array);
13    for (i = 0; i < n; i++)
14      array[i]++;
15    return;
16  }
```

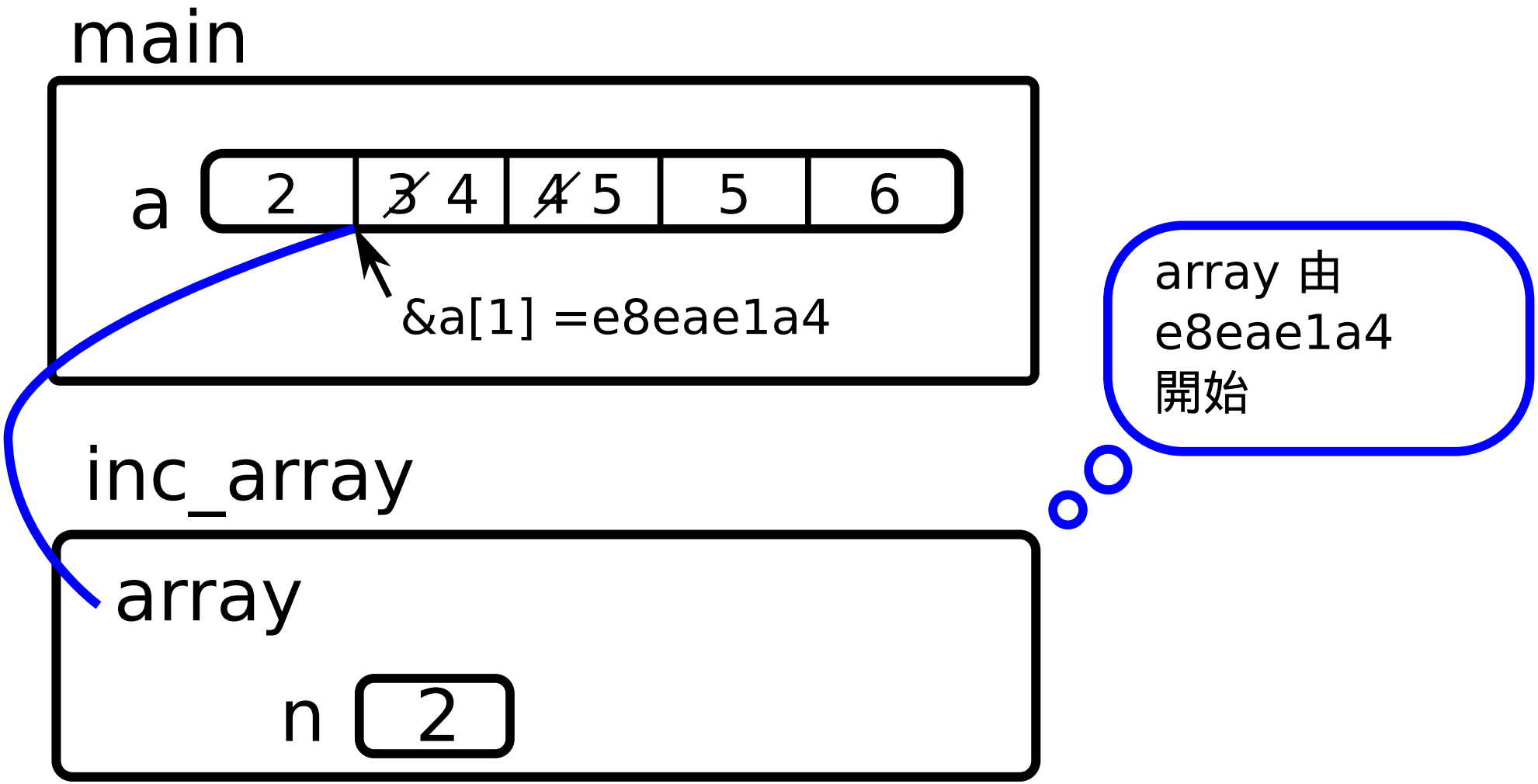
```
18 int main(void)
19 {
20   int i;
21   int a[5];
22   for (i = 0; i < 5; i++)
23     scanf("%d", &(a[i]));
24   printf("before inc_array\n");
25   print_array(a, 5);
26   inc_array(a, 5);
27   printf("after first inc_array\n");
28   print_array(a, 5);
29   inc_array(&(a[1]), 2);
30   printf("after second inc_array\n");
31   print_array(a, 5);
32   inc_array(&(a[2]), 2);
33   printf("after second inc_array\n");
34   print_array(a, 5);
35   return 0;
36 }
```

1. 第一次實際參數是 (array, 5)，所以 inc_array 中的形式參數 a 會拿到陣列 array 的起始位址，並將整個陣列加 1。
2. 第二次實際參數是 (&(array[1]), 2)，所以 inc_array 中的形式參數 a 會拿到元素 array[1] 的位址，並將 array[1] 及 array[2] 加 1。
3. 第三次實際參數是 (&(array[2]), 2)，所以 inc_array 中的形式參數 a 會拿到元素 array[2] 的位址，並將 array[2] 及 array[3] 加 1。

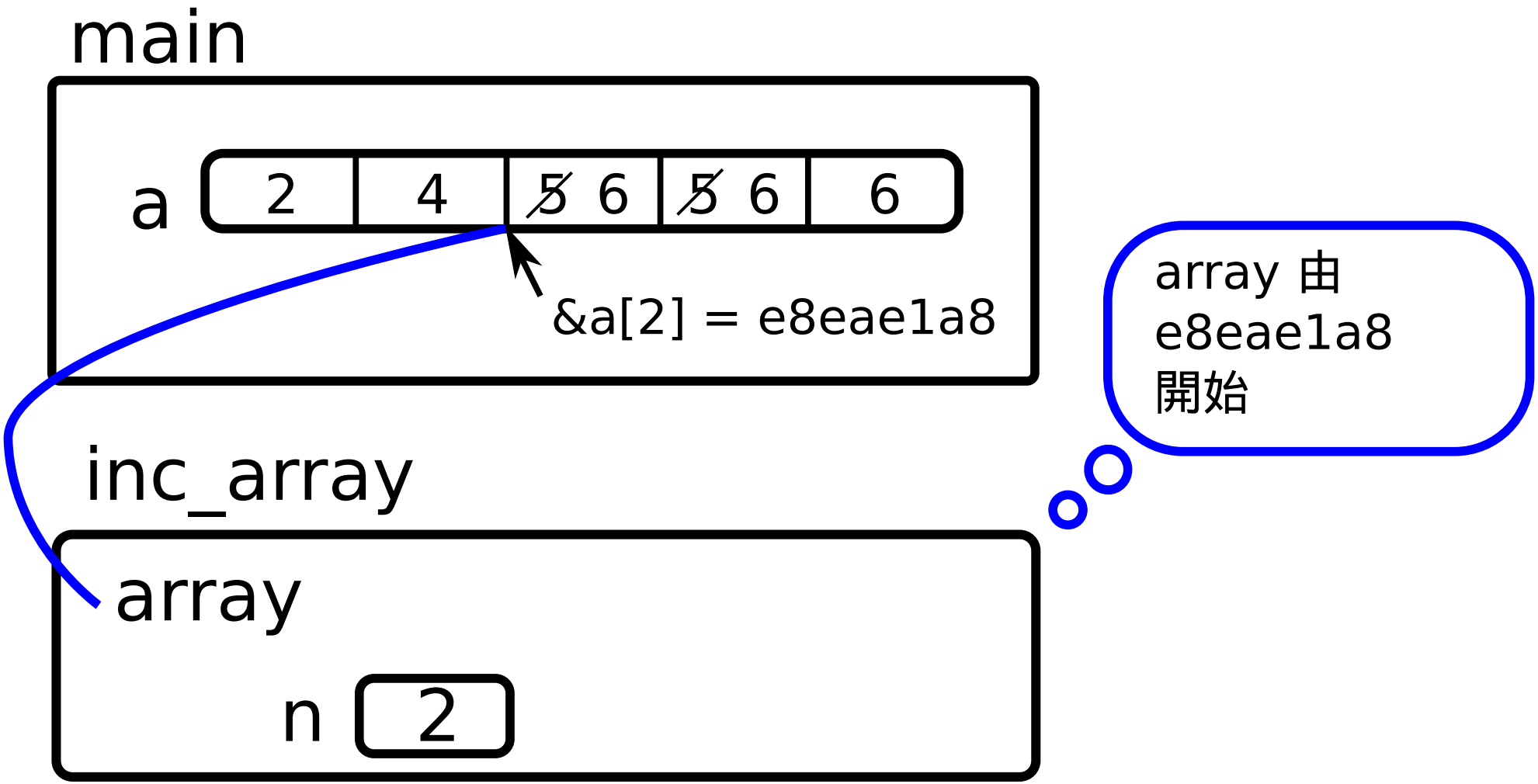
傳遞陣列


main
a
1 2
2 3
3 4
4 5
5 6
a = e8eae1a0
array 由
a = e8eae1a0
開始
inc_array
array
n
5

傳遞陣列


main
a
2
3 4
4 5
5
6
&a[1] =e8eae1a4
array 由
e8eae1a4
開始
inc_array
array
n
2

傳遞陣列


main
a
2
4
~~5~~ 6
~~5~~ 6
6
&a[2] = e8eae1a8
array 由 e8eae1a8 開始
inc_array
array
n
2

範例程式 27: (multi-dim-array-parameter.c) 傳遞多維陣列參數

```
1  #include <stdio.h>
2  void print_matrix(int a[4][3], int i, int j)
3  {
4    printf("a[%d][%d] = %d\n", i, j, a[i][j]);
5    return;
6  }
7  int main(void)
8  {
9    int i, j;
10   int array[3][4];
11   for (i = 0; i < 3; i++)
12     for (j = 0; j < 4; j++)
13       scanf("%d", &(array[i][j]));
14   printf("array[2][1] = %d\n", array[2][1]);
15   print_matrix(array, 2, 1);
16   printf("array[0][2] = %d\n", array[0][2]);
17   print_matrix(array, 0, 2);
18   return 0;
19 }
```

- 傳遞多維陣列參數時，必須注意形式參數及實際參數的一致性。
- 形式參數 a 的宣告是 [4][3]，而實際參數 array 的宣告是 [3][4]。
- 第 12 行 array[2][1] 在計算位址時，得到 $2 \times 4 + 1 = 9$，所以會印出 9。
- 而第 4 行在計算 a[2][1] 的位址時，因為 a 的宣告是 [4][3]所以會得到 $2 \times 3 + 1 = 7$，並且印出 7。

輸入

```
1  0  1  2  3
2  4  5  6  7
3  8  9  10  11
```

輸出

```
1  array[2][1] = 9
2  a[2][1] = 7
3  array[0][2] = 2
4  a[0][2] = 2
```

傳遞多維陣列

in memory

0	1	2	3	4	5	6	7	8	9	10	11

	0	1	2	3
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11

main

	0	1	2
0	0	1	2
1	3	4	5
2	6	7	8
3	9	10	11

print_matrix