資料結構 C 演算法

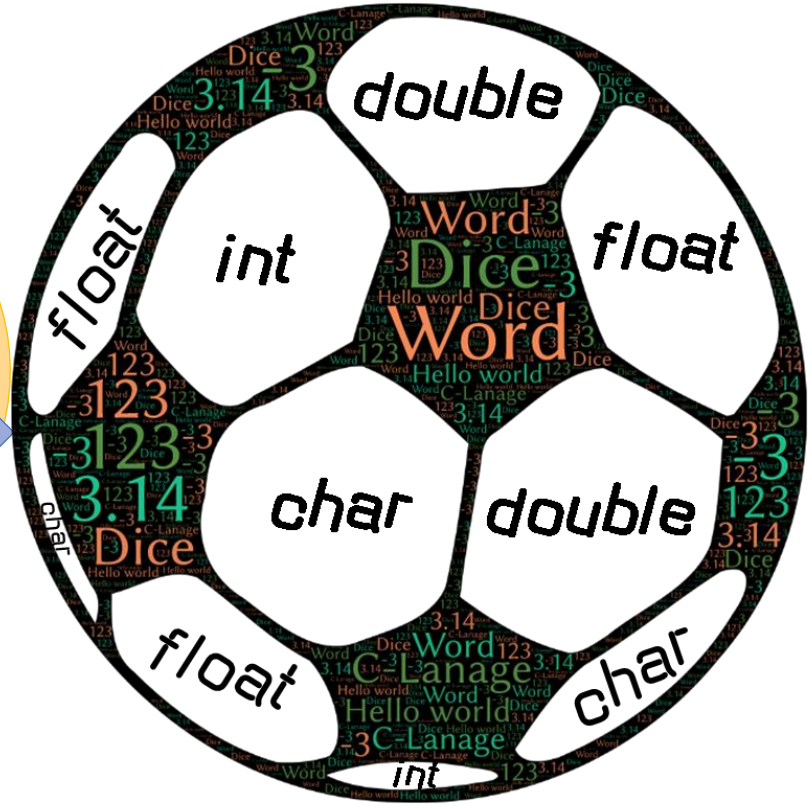
指標 Pointer

指標 Pointer


什麼是指標？

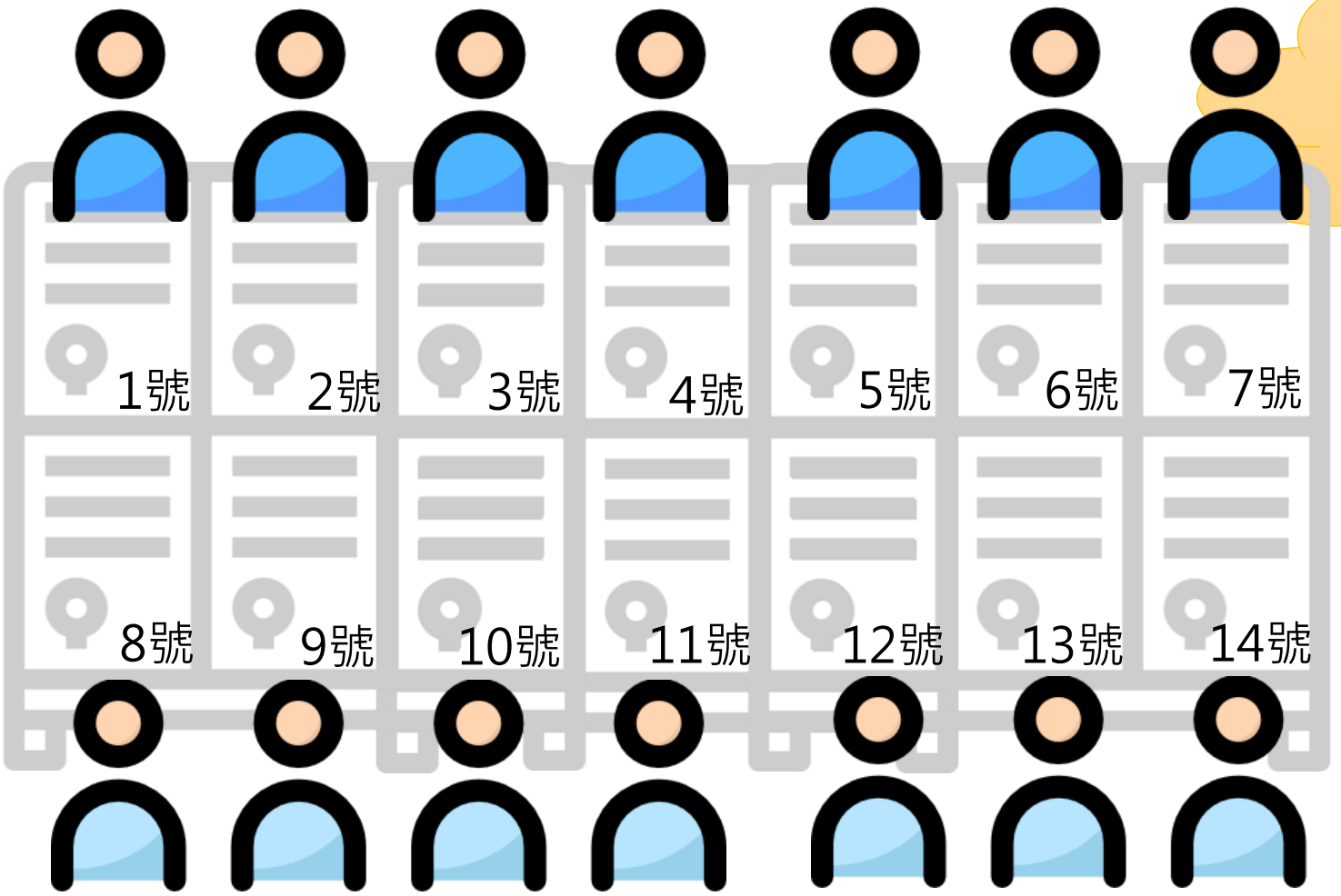
還記得資料是怎麼儲存的嗎？

1. 向電腦申請一個櫃子準備用來儲放資料
2. 將資料放進櫃子裡面的抽屜
3. 透過一個**變數名稱**取得抽體裡的資料內容



要怎麼知道哪個變數名稱
是對應哪個抽屜呢？

先想想這個情況

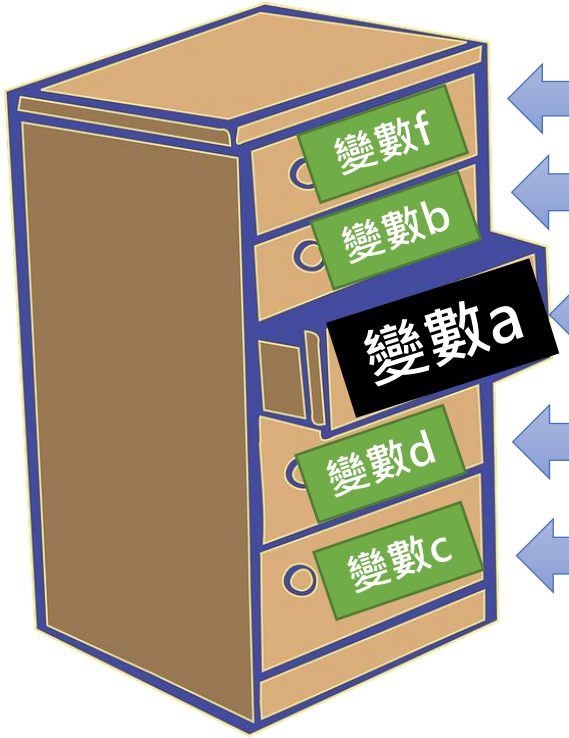


班級裡每個人都有自己的櫃子，班級內的學生都會知道哪個學生是對應哪個櫃子。

假若讓一個非班級內的路人甲去打開 3 號學生的櫃子，他會知道到底是哪個櫃子嗎？


讓路人甲打開
櫃子上層第一排，
左邊數來第三個位置
的櫃子就好啦！

同樣的...



第 1 格

第 2 格

第 3 格

第 4 格

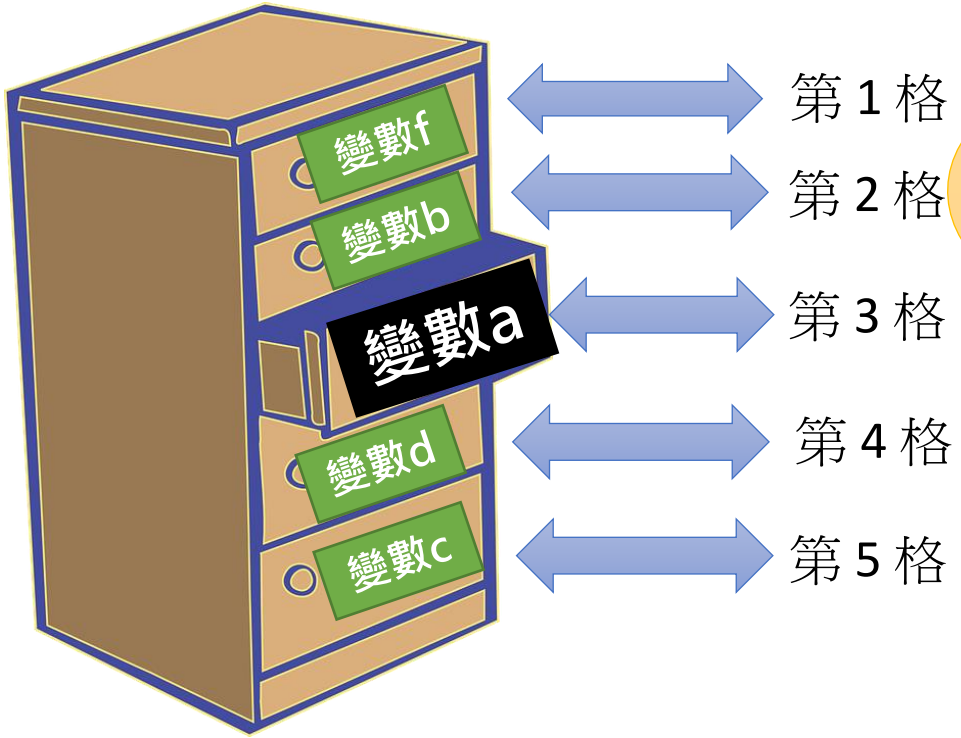
第 5 格

取 **變數f** 資料就
是要打開**第 1 格**

取 **變數a** 資料就
是要打開**第 3 格**

取 **變數c** 資料就
是要打開**第 5 格**

這就是『指標』



資料實際對應的位置
(位址)

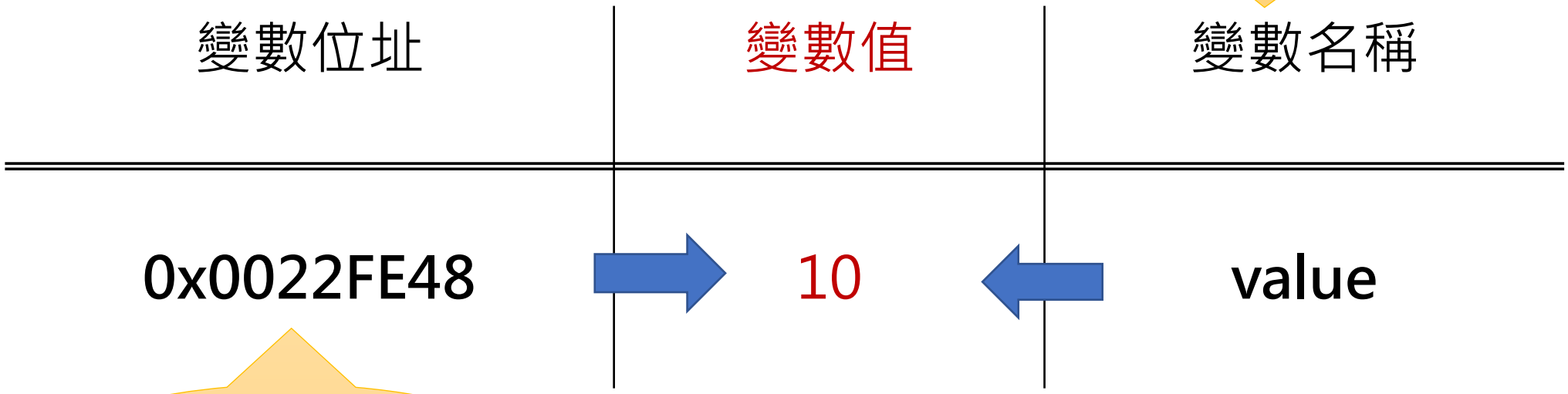
就是『指標』


指標與變數

變數三大元素

```
Int value = 10;
```

變數名稱是提供給**程式編輯者**看



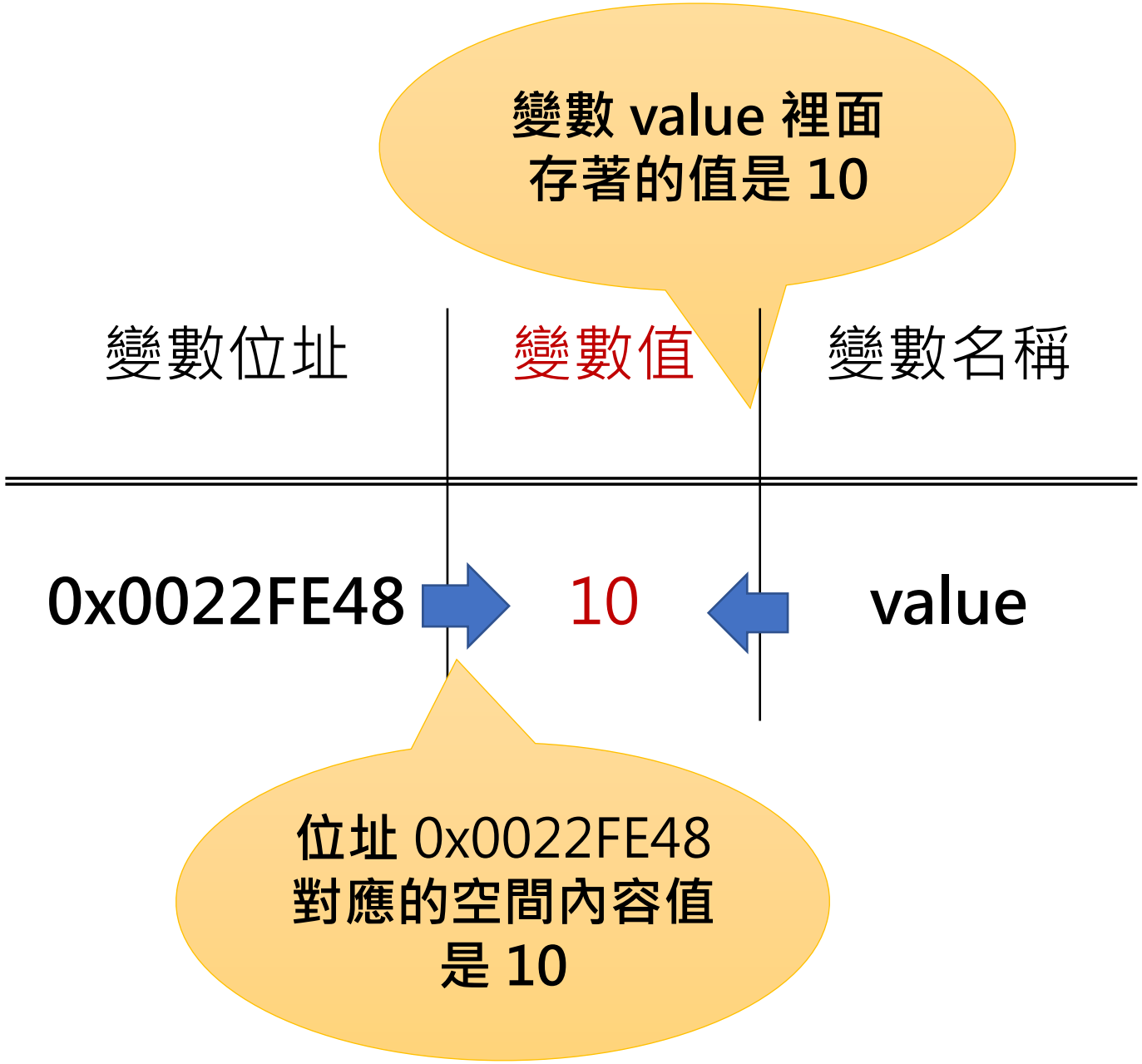
變數位址是**程式內部執行**時看的

變數的產生

```c
#include <stdio.h>
int main()
{
  int value;

  value = 10;

  return 0;
}
```



變數的產生流程: 第 1 步

```c
#include <stdio.h>
int main()
{
  int value;

  value = 10;

  return 0;
}
```

第 1 步

從閒置的記憶體內找出符合變數型態所需大小的空間，得到這個閒置空間的位址

0x0022FE43	0x0022FE44	0x0022FE45	0x0022FE46	0x0022FE47	0x0022FE48
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte

4 bytes
(int 需要 4 個 bytes 的空間)

變數的產生流程: 第 2 步

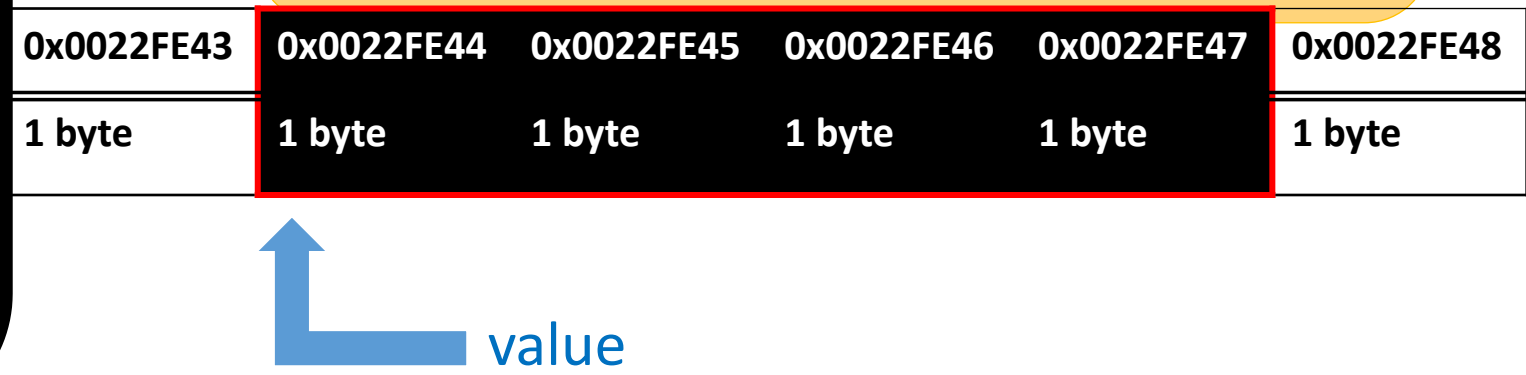
```c
#include <stdio.h>
int main()
{
  int value;

  value = 10;

  return 0;
}
```

第 2 步

將宣告的變數名稱對應找到的閒置空間的位址



變數的產生流程: 第 3 步

```c
#include <stdio.h>
int main()
{
  int value;

  value = 10;

  return 0;
}
```

第 3 步

將指定的變數值放到變數名稱對應的空間內

0x0022FE43	0x0022FE44	0x0022FE45	0x0022FE46	0x0022FE47	0x0022FE48
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte
	10				

value

要如何取得變數是放在哪個空間位址呢？

取得變數的空間位址

```c
#include <stdio.h>
int main()
{
  int value;
  int *ptr;
  ptr = &value;
  value = 10;
  printf("value: %d\n", value);
  printf("value位址: %p\n", &value);
  printf("ptr: %p\n", ptr);
  printf("ptr對應內容值: %d\n", *ptr);
  return 0;
}
```

定義指標變數 ptr，透過**取址運算符號(&)**取得指定變數的位址，將取得的位址交給指標變數 ptr

執行結果

```
dice - pointer $ gcc 02.c -o 02.exe
dice - pointer $ ./02.exe
value: 10
value位址: 000000000022FE44
ptr: 000000000022FE44
ptr對應內容值: 10
```

取得變數的空間位址

02.c

```c
#include <stdio.h>
int main()
{
  int value;
  int *ptr;
  ptr = &value;
  value = 10;
  printf("value: %d\n", value);
  printf("value位址: %p\n", &value);
  printf("ptr: %p\n", ptr);
  printf("ptr對應內容值: %d\n", *ptr);
return 0;
}
```

概念1: 取址運算符號 &

概念2: 位址輸出 %p

概念3: 指標變數

概念4: 取值運算符號 *

取得變數的空間位址

概念1: 取址運算符號 &

```c
#include <stdio.h>
int main()
{
  int value;
  int key;
  char ch;

  printf("value位址: %p\n", &value);
  printf("key位址: %p\n", &key);
  printf("ch位址: %p\n", &ch);
  return 0;
}
```

& 稱為**取址運算子** (Address-of operator)，會取得所接變數名稱的位址

取得變數的空間位址

概念2: 位址輸出 %p

```c
#include <stdio.h>
int main()
{
  int value;
  int key;
  char ch;

  printf("value位址: %p\n", &value);
  printf("key位址: %p\n", &key);
  printf("ch位址: %p\n", &ch);
  return 0;
}
```

輸出時使用 %p ，就會輸出位址資料

%p 輸出的格式會跟 %x 輸出的格式一樣，都是十六進制的資料，但為了閱讀認知方便，仍建議輸出位址時使用 %p

取得變數的空間位址

概念3: 指標變數

每個變數都可以透過取址符號(&)取得對應的記憶體位址

```c
#include <stdio.h>
int main()
{
  int intVar1 = 10, intVar2 = 20, intVar3 = 30;
  char charVar1 = 's', charVar2 = 'A';
  int *intVarPointer1 = &intVar1;
  int *intVarPointer2, *intVarPointer3;
  char *charVarPointer1 = &charVar1, *charVarPointer2 = &charVar2;
  intVarPointer2 = &intVar2;
  intVarPointer3 = &intVar3;
  charVarPointer2 = &charVar2;
  printf("intVar1 的值: %d, 位址: %p\n", intVar1, intVarPointer1);
  printf("intVar2 的值: %d, 位址: %p\n", intVar2, intVarPointer2);
  printf("intVar3 的值: %d, 位址: %p\n", intVar3, intVarPointer3);
  printf("charVar1 的值: %c, 位址: %p\n", charVar1, charVarPointer1);
  printf("charVar2 的值: %c, 位址: %p\n", charVar2, charVarPointer2);
  return 0;
}
```

執行結果

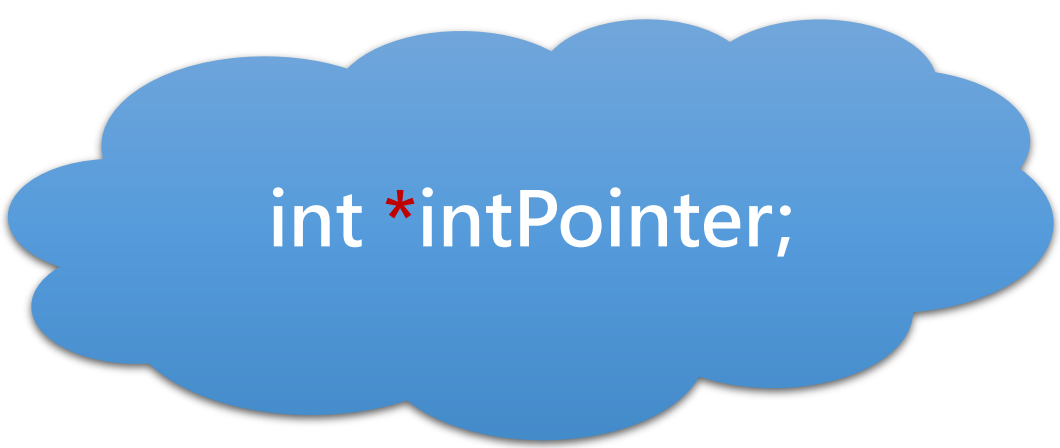
```
dice - pointer $ ./04.exe
intVar1 的值: 10, 位址: 000000000022FE48
intVar2 的值: 20, 位址: 000000000022FE44
intVar3 的值: 30, 位址: 000000000022FE40
charVar1 的值: s, 位址: 000000000022FE3F
charVar2 的值: A, 位址: 000000000022FE3E
```

取得變數的空間位址

概念3: 指標變數(1): 宣告

- 與一般變數相同點：
  - 要使用指標變數前要先宣告
  - 宣告方式也與一般變數相似
  - **要指向什麼型態的變數資料位址，就使用相同的資料型態宣告指標變數。**

- 唯一的差異只在於：**宣告指標變數時，前面需要加上星號(*)**



取得變數的空間位址

概念3: 指標變數(2): 給值

取得變數的空間位址

概念4: 取值運算符號 *

```c
#include <stdio.h>
int main()
{
   int value;
   int *ptr;
   ptr = &value;
   value = 10;
   printf("value: %d\n", value);
   printf("value位址: %p\n", &value);
   printf("ptr: %p\n", ptr);

   printf("ptr對應內容值: %d\n", *ptr);
   printf("value對應內容值: %d\n", *&value);
   return 0;
}
```

* 稱為**取值運算子 (Dereference Operator)**，會取得所接位址的儲存值

取得變數的空間位址

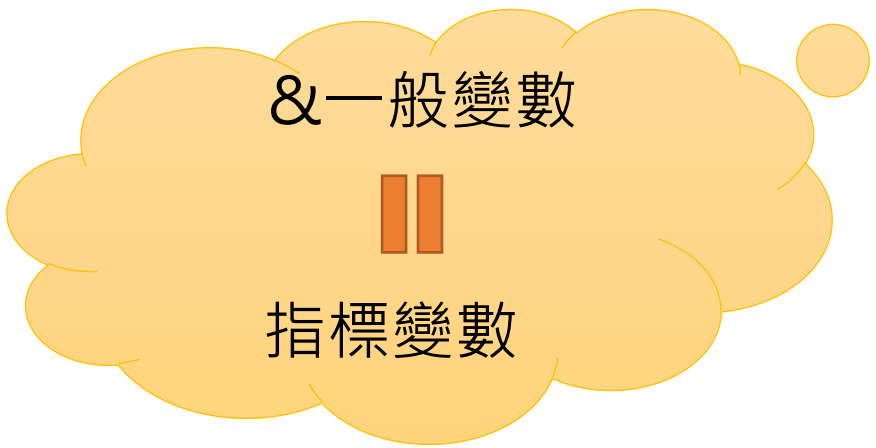
概念4: 取值運算符號 *

取址運算符號 &

- 一般變數透過取址符號(&)得到資料位址
- 格式: &一般變數

取值運算符號 *

- 資料位址透過取值符號(*)得到資料內容
- 格式: *空間位址
  - *指標變數
  - *(&一般變數)



請記得

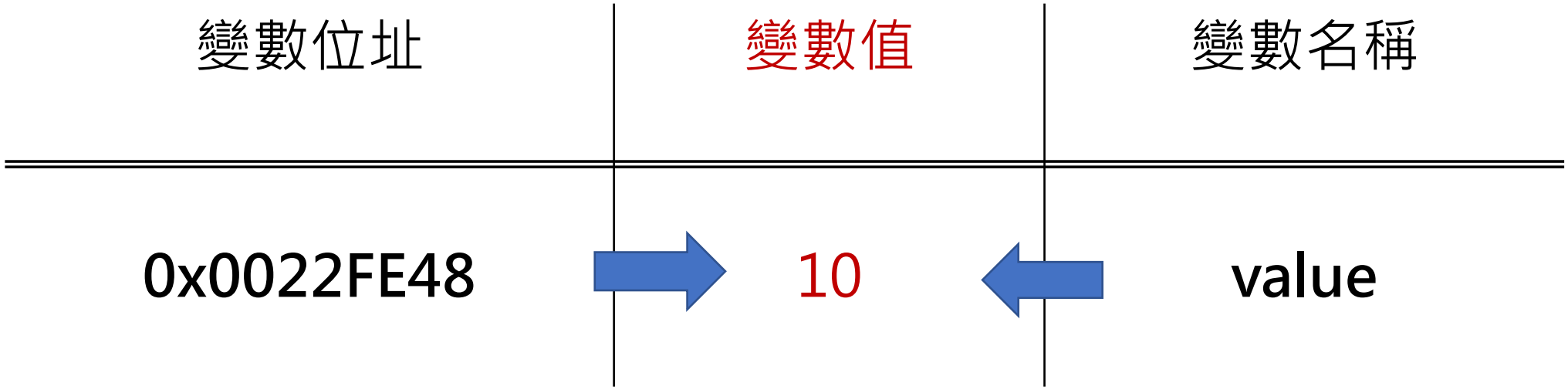
變數三大元素

- 變數位址
- 變數值
- 變數名稱

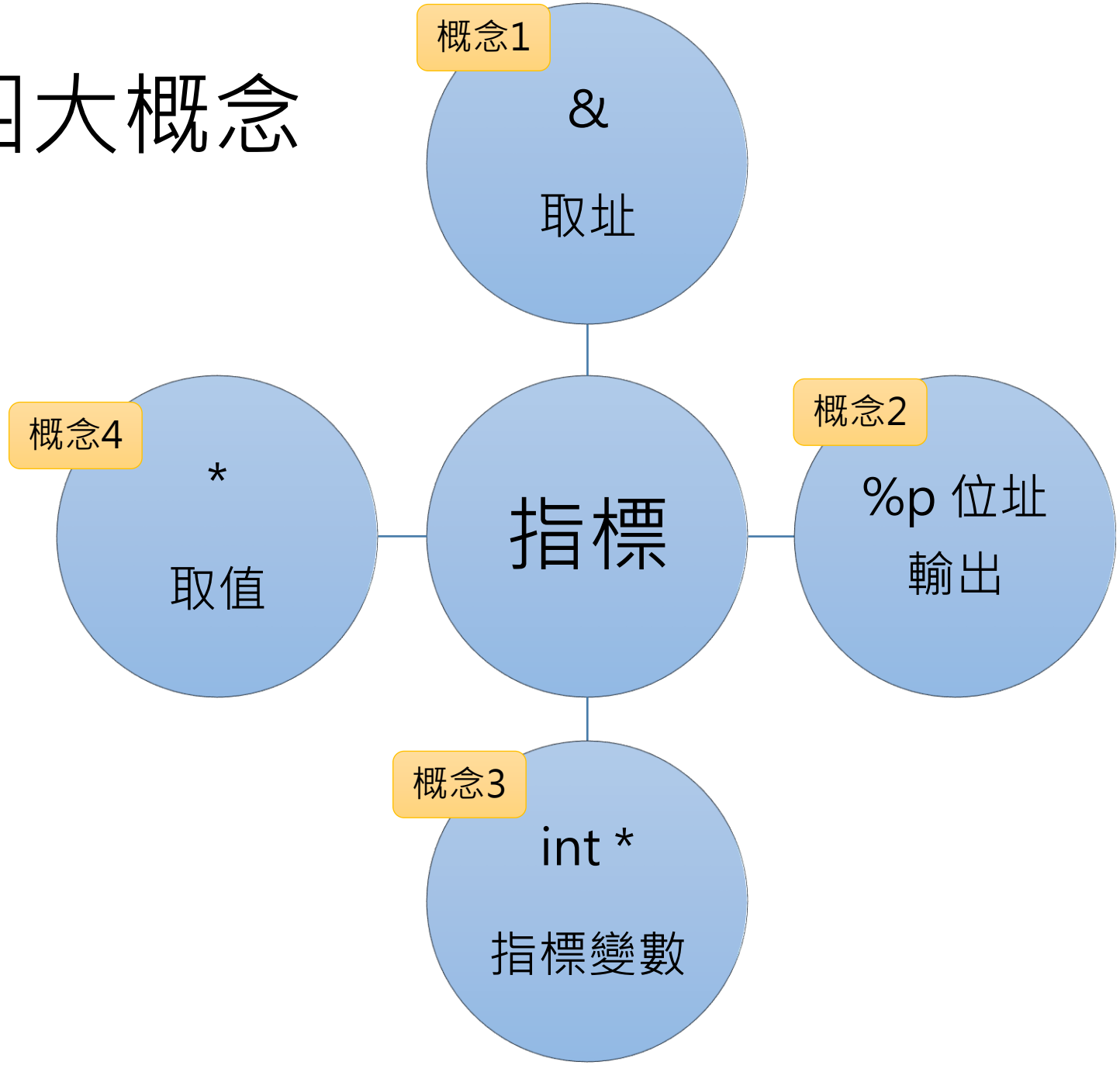
指標四大概念

- 取址運算符號 &
- 位址輸出 %p
- 指標變數
- 取值運算符號 *

變數三大元素 Int value = 10;



指標四大概念



為什麼要用指標？

沒有指標時的困境(1)

```c
#include <stdio.h>
int sub(int a, int b)
{
  a = a + 10;
  b = b - 10;
  return a;
}
int main()
{
  int a = 25, b = 100;
  printf("[呼叫前] a = %d, b = %d\n", a, b);
  sub(a, b);
  printf("[呼叫後] a = %d, b = %d\n", a, b);
  return 0;
}
```

函式只能用 return 回傳一個值，如果傳入的 2 個參數都要改值後再回傳呢？

沒有指標時的困境(2)

```c
#include <stdio.h>
int main()
{
  int num = 10;
  int num2 = 0;
  int data[num];
  int i = 0;
  scanf("%d", &num2);
  printf("num2: %d\n", num2);
  for (i = 0; i < num; i++)
  {
    data[i] = i;
    printf("data[%d]: %d\n", i, data[i]);
  }
  return 0;
}
```

如果陣列 data 的大小要由 num2 決定呢？

num2 的值會在陣列宣告後才取得，要怎麼才能之後再動態決定陣列大小呢？

指標是解藥也是毒藥

- 指標可以用來解決 C 程式很多難題，諸如記憶體大小配置、資料傳送處理等問題。
  - 後面會繼續說明指標的各個不同用法

- 但若不小心將指標指錯位置，那可能會造成程式後續執行時有很多資料處裡發生大錯誤，因此使用指標務必要小心謹慎！


指標與常數

指標與常數有三種搭配

- **指向常數的指標**(A Pointer to a Constant)
  - const int *ptr
- **常數指標**(A Constant Pointer)
  - int *const ptr = &intVar
- **指向常數的常數指標**(A Constant Pointer to a Constant)
  - const int *const ptr = &intVal

指向常數的指標
A Pointer to a Constant

```
const int  *ptr;
```

- 指向儲存常數資料的位置
- 也就是代表指標指向的位址內的值是常數，**位址內的值是不能被更動的**！
- 儲存的位址是可以更動的！

常數指標
A Constant Pointer

```
int *const ptr = &intVar;
```

- 指標本身是常數值
- 一開始指派給指標的位址是常數，因此之後**不可更改記錄的位址**。
- 位址內的值不是常數，仍可更改。

指向常數的常數指標
A Constant Pointer to a Constant

```
const int *const ptr = &intVal;
```

- 指向常數的指標 + 常數指標
- 因此**位址內的值是不能被更動的**！
- 而且**不可更改記錄的位址**。

	指標對應的位置內容值	指標存的位址	格式
指向常數的指標 A Pointer to a Constant	不可更改	可以更改	const 型態* 變數名稱
常數指標 A Constant Pointer	可以更改	不可更改	型態* const 指標變數名稱 = &一般變數名稱
指向常數的常數指標 A Constant Pointer to a Constant	不可更改	不可更改	const 型態* const 指標變數名稱 = &一般變數名稱


指標的運算

整數的運算

```c
#include <stdio.h>
int main()
{
  int value = 10;
  printf("value = %d\n", value);
  value = value + 1;
  printf("value + 1 = %d\n", value);
  value = value + 1;
  printf("value + 2 = %d\n", value);
  return 0;
}
```

執行結果

```
dice - pointer $ ./08.exe
value = 10
value + 1 = 11
value + 2 = 12
```

整數有加減乘除等算術運算

指標格式如同十六進位的整數，是否也會有同樣的加減乘除等算術運算？

指標的算術運算

字元指標的運算

```c
#include <stdio.h>
int main()
{
  char value;
  char *ptr = &value;
  printf("ptr: %p\n", ptr);
  ptr = ptr + 1;
  printf("ptr + 1 = %p\n", ptr);
  ptr = ptr + 1;
  printf("ptr + 2 = %p\n", ptr);
  return 0;
}
```

好像跟十六進位整數運算相同？

22FE4B + 1 = 22FE4C

22FE4B + 2 = 22FE4D

執行結果

```
dice - pointer $ ./09.exe
ptr: 000000000022FE4B
ptr + 1 = 000000000022FE4C
ptr + 2 = 000000000022FE4D
```

整數指標的運算

```c
#include <stdio.h>
int main()
{
  int value;
  int *ptr = &value;
  printf("ptr: %p\n", ptr);
  ptr = ptr + 1;
  printf("ptr + 1 = %p\n", ptr);
  ptr = ptr + 1;
  printf("ptr + 2 = %p\n", ptr);
  return 0;
}
```

疑，十六進位的整數 22FE48
的加法應該是:
22FE48 + 1 = 22FE49
22FE48 + 2 = 22FE4A

執行結果

```
dice - pointer $ ./10.exe
ptr: 000000000022FE48
ptr + 1 = 000000000022FE4C
ptr + 2 = 000000000022FE50
```

指標的運算

- 指標加減法中，每加(減)1是代表前進(後退)一個記憶體長度單位，而記憶體長度單位是幾個位元組會依照指標的資料型態而決定。

- 字元型態:
  - 字元指標 + 1 => 指標位址 + 1 byte
  - 字元指標 - 1 => 指標位址 - 1 byte

- 整數指標:
  - 字元指標 + 1=> 指標位址 + 4 bytes
  - 字元指標 - 1 => 指標位址 - 4bytes

各種資料型態存放的記憶體長度單位可使用函式 **sizeof()** 取得，因此:
指標變數+1=>指標變數 + sizeof(型態)

指標與指標的相減

同樣的，指標與指標相減後的值也是使用指標資料型態的記憶體長度做為基本單位。

注意: 指標與指標是不能相加的喔！

```c
#include <stdio.h>
int main()
{
  int value1, value2;
  int *ptr1 = &value1, *ptr2 = &value2;
  printf("ptr1: %p\n", ptr1);
  printf("ptr2: %p\n", ptr2);
  printf("ptr1 - ptr2 = %d\n", ptr1 - ptr2);
  return 0;
}
```

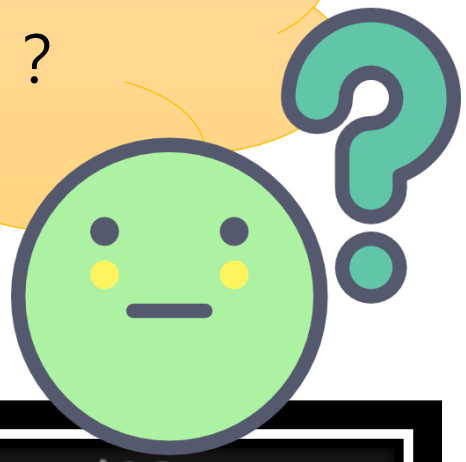
執行結果

```
dice - pointer $ ./11.exe
ptr1: 000000000022FE48
ptr2: 000000000022FE44
ptr1 - ptr2 = 1
```

但是，指標為什麼要做加減法？

```c
#include <stdio.h>
int main()
{
  int value;
  int *ptr = &value;
  printf("ptr: %p\n", ptr);
  ptr = ptr + 1;
  printf("ptr + 1 = %p\n", ptr);
  ptr = ptr + 1;
  printf("ptr + 2 = %p\n", ptr);
  return 0;
}
```

這個運算有意義嗎？



執行結果

```
dice - pointer $ ./10.exe
ptr: 000000000022FE48
ptr + 1 = 000000000022FE4C
ptr + 2 = 000000000022FE50
```

指標的運算

- 一個指向一般變數的指標進行加減法運算是沒意義的。
  - 無法保證任何變數儲存在記憶體中的位址，即使同時宣告的兩個變數也無法保證位址就會相鄰，因此將指標加減法後得到往前或往後幾個資料型態單位的記憶體位址是沒意義的。

- 只有指向**陣列**變數的指標進行加減法運算才有意義喔！

- 因此，請先記著「**指標的加(減)1是代表前進(後退)一個記憶體長度單位**」的概念，此概念將會是下一個主題「**指標與陣列**」的重點之一。


指標與函式

前面提過的「沒有指標時的困境(1)」

```c
#include <stdio.h>
int sub(int a, int b)
{
  a = a + 10;
  b = b - 10;
  return a;
}
int main()
{
  int a = 25, b = 100;
  printf("[呼叫前] a = %d, b = %d\n", a, b);
  sub(a, b);
  printf("[呼叫後] a = %d, b = %d\n", a, b);
  return 0;
}
```

函式只能用 return 回傳一個值，如果傳入的 2 個參數都要改值後再回傳呢？

函式的參數: 傳值(call by value)

```
#include <stdio.h>
int sub(int a, int b)
{
}

int main()
{
   sub(a, b);
```

sub() 內的變數資訊

變數名	變數值	變數位址
a	10	0x0022FF58
b	20	0x0022FE54

main() 內的變數資訊

變數名	變數值	變數位址
a	10	0x0022FE48
b	20	0x0022FE44

只有將變數值傳入，但傳入後會將資料另外放在別的記憶體位址，因此兩邊的**變數位址是不同**的！

函式的參數: 傳值(call by value)

- 使用一般變數做為參數傳遞給函式時，只有傳變數的內容值給函式，此種方式稱為傳值法(call by value)。
  - 函式: int sub(int a, int b)
  - 在 sub() 函式內會有另一個記憶體空間存放從外部傳入的 a 與 b 的內容值，因此即使在函式內有更改 a 與 b 的值，呼叫者那邊也無法接收到。

- 使用傳值法傳入的變數，若要讓呼叫者拿到變數新的值(在函式內更改後的值)，就只能透過函式的 return 回傳，因此會受 return 只能回傳一個值的限制影響。

函式的參數: 傳址(call by address)

- 不使用一般變數，改用指標變數做為函式的參數。
  - 格式: type sub(type *a, type *b)

- 由於函式的參數是使用指標型態，因此呼叫函式時傳入的格式必須使用位址格式。
  - 格式: **result = sub(&a, &b)**

- 參數傳入的是記憶體位址，函式內只要搭配取值運算符號，就可直接更改原呼叫區內的變數值。
  - 格式: ***指標變數名 = 新值**

函式的參數: 傳址(call by address)

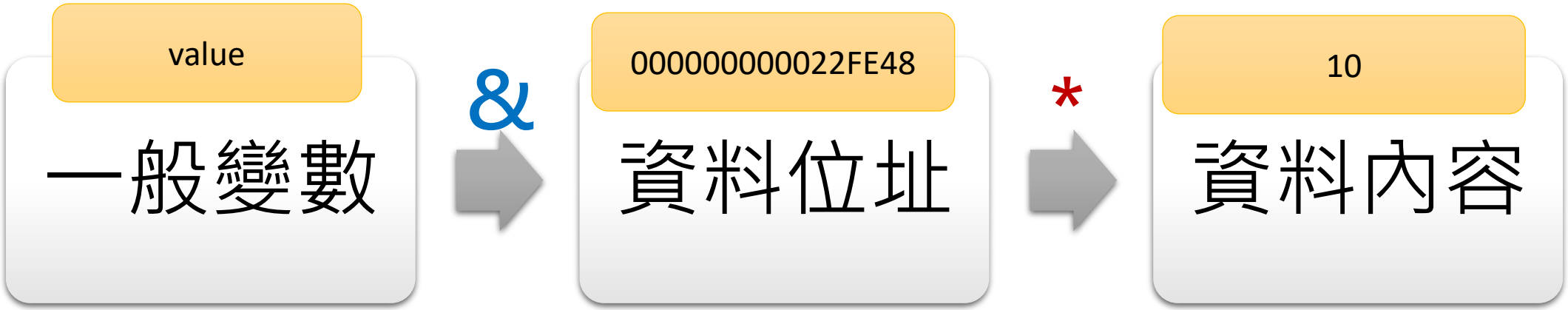
```c
#include <stdio.h>
void sub(int *a, int *b)
{
  *a = *a + 10;
  *b = *b - 10;
}
int main()
{
  int a = 25, b = 100;
  printf("[呼叫前] a = %d, b = %d\n", a, b);
  sub(&a, &b);
  printf("[呼叫後] a = %d, b = %d\n", a, b);
  return 0;
}
```

函式的參數

	傳值(call by value)	傳址(call by address)
函式呼叫	func(a, b)	func(&a, &b)
函式參數接收	int func(int a, int b)	int func(int *a, int *b)
函式內改值	a = a+10 搭配 return a	*a = *a + 10

延伸的概念

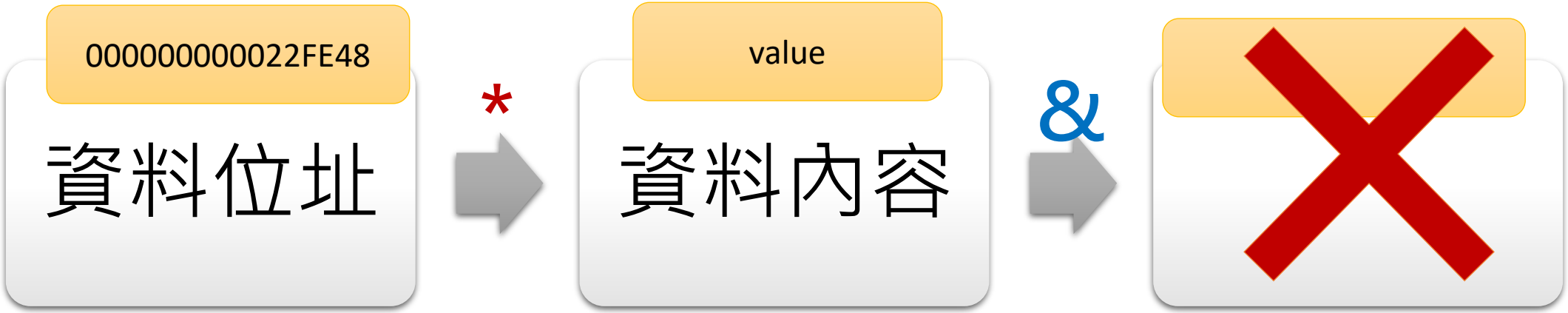
概念1: 取址 + 取值 v.s 取值 + 取址



13.c

```c
#include <stdio.h>
int main()
{
  int value = 10;
  int *ptr = value;
  printf("value: %d\n", value);
  printf("value位址: %p\n", &value);
  printf("value對應內容值: %d\n", *&value);
  return 0;
}
```

概念1: 取址 + 取值 v.s 取值 + 取址



資料內容值本身是沒有位址的，
儲存該內容值的變數才有位址！

概念2: 函式指標(function pointer)

- 函式指標: 指向函式的位址的指標

函式回傳型態 (*指標名稱)(函式參數型態)

```c
#include <stdio.h>
int getFirst(int a, int b)
{
  return a;
}
int getLast(int a, int b)
{
  return b;
}
int main()
{
  int result;
  int (*funcPointer)(int, int);
  funcPointer = getFirst;
  result = funcPointer(10, 20);
  printf("getFirst result: %d\n", result);
  funcPointer = getLast;
  result = funcPointer(10, 20);
  printf("getLast result: %d\n", result);
  return 0;
}
```

函式指標的資料型態需要與函式的回傳型態一致！

funcPointer 是函式指標 ，會指向一個需要傳入兩個整數參數的函式

函式名稱本身會記錄函式的起始位址

概念2: 函式指標(function pointer)

- 函式指標的主要用途是讓函式可做為另一個函式的參數傳入。
  - int **func**(int, int, **int (*ptr)(int, int)**)

func 有三個參數：
1. 整數變數
2. 整數變數
3. 函式指標

func 的第三個參數為函式指標，是只像一個需要傳入兩個整數參數且會回傳整數值的函式

```c
#include <stdio.h>
int getMin(int a, int b)
{
  return a > b ? b : a;
}
int getMax(int a, int b)
{
  return a > b ? a : b;
}
int compare(int a, int b, int (*cmp)(int, int))
{
  return cmp(a, b);
}
int main()
{
  int result;
  result = compare(10, 20, getMax);
  printf("result: %d\n", result);
  result = compare(10, 20, getMin);
  printf("result: %d\n", result);
  return 0;
}
```

傳入一個函式指標

重點整理

- 什麼是指標？
- 變數三大元素：變數位址、變數值、變數名稱
- 指標四大概念：取址運算符號 & 、位址輸出 %p 、指標變數、取值運算符號 *
- 指標與常數
- 指標的運算
- 指標與函式：call by value、 call by address
- 函式指標