

C++程式設計基礎

lesson 6

陳毅

a693: 吞食天地

演算法
Dynamic Programming (動態規劃)

內容

好餓歐歐歐歐

有 n 個食物在你面前排成一排

每個食物有它的飽足度

你想知道把其中一段通通吃掉會獲得多少飽足度

輸入說明

多組測資以 EOF 結束

每組測資開始有兩個正整數 n, m ($n, m \leq 100000$)

接下來一行有 n 個不超過一千的正整數依序代表每個食物的飽足度

接下來 m 行每行有兩個數字 l, r ($1 \leq l \leq r \leq n$)

代表你想要吃掉第 l 個到第 r 個食物

輸出說明

對每組測資輸出 m 行，代表總飽足度

範例輸入

```
3 3
1 2 3
1 3
1 2
2 3
```

範例輸出

```
6
3
5
```

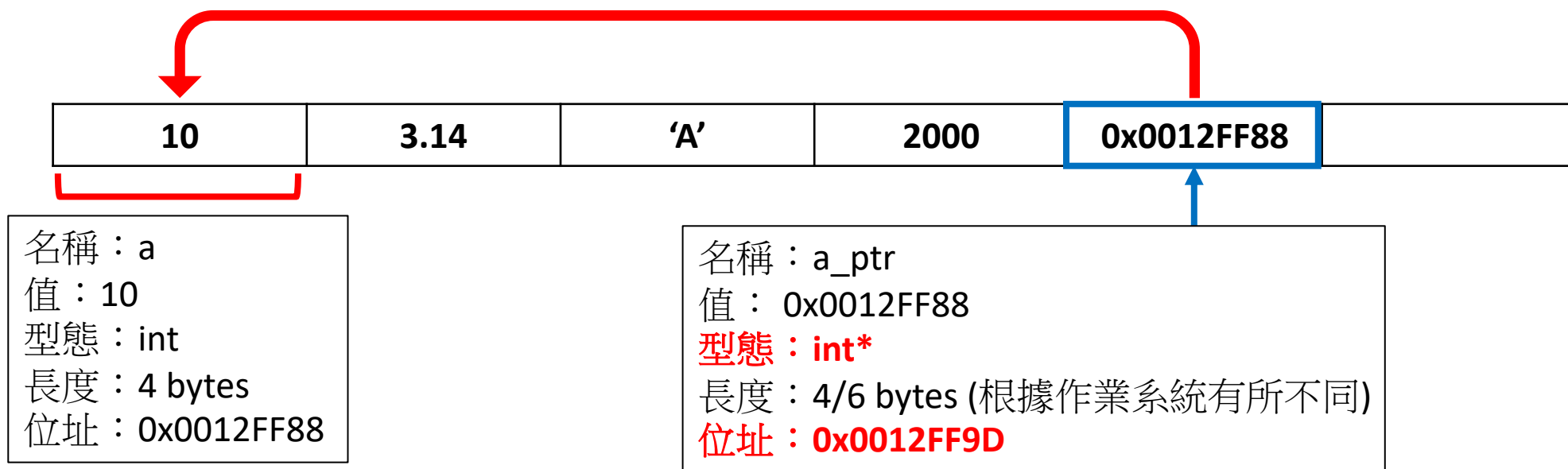
上節回顧

- 宣告一個用來儲存「記憶體位址」的變數，此變數稱作指標。
- 使用「位址運算符 &」得到一個變數的記憶體位址。
- 使用「間接運算符 *」得到一個記憶體位址所儲存的值。
- 陣列的指標
 - 指標運算
 - 多重指標
- 使用指標作為函數的引數（參數），來傳遞資料。

指標 (Pointer)

- 指標是一種資料型態，用來儲存記憶體位址。
- 指標本身也具有記憶體空間。
- 變數型態：int*, float*, int**, double**等。

```
int *a_ptr = &a;
```



宣告指標

資料型態 *指標變數;

- 宣告指標變數與宣告一般變數的方法類似，只是在指標變數前面加上「*」或是在資料型態後面加上「*」。

容易在同時宣告多個變數時出現誤用

誤用

- 宣告兩個指標變數ptr1與ptr2。

```
int* ptr1, ptr2;
```

ptr1為指標變數
ptr2為int變數

```
int *ptr1, *ptr2;
```

正確

指標與變數的參考

- 取得變數的**記憶體位址**
 - 「&」稱作位址運算符(**address-of operator**)，是用來取得變數的位址，也稱作**參考運算符號(reference operator)**。

&變數名稱

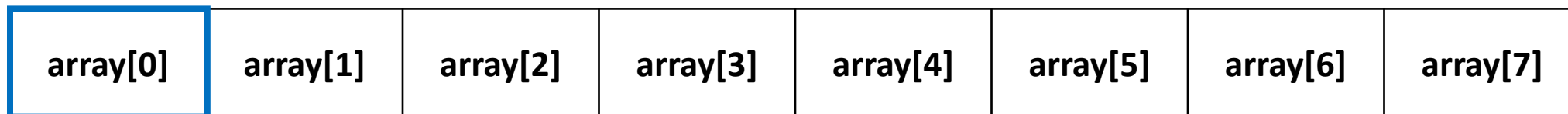
- 取得一個記憶體位址所儲存的**值**
 - 「*」稱作間接運算符號(**indirect operator**)，是用來取得參考位址內的值，也稱作**反參考運算符號(de-reference operator)**。

***指標名稱**

陣列的指標

- 一維陣列（以int[8]為例）

$\text{array} + 2 == \&\text{array}[2]$



記憶體位址：0x7ffee46cdac0

$\text{array}[i] = \text{array}[0] + i * \text{sizeof}(\text{int})$
一個int佔用4個bytes

第0個元素的位址：	0x7ffee46cdac0
第1個元素的位址：	0x7ffee46cdac4
第2個元素的位址：	0x7ffee46cdac8
第3個元素的位址：	0x7ffee46cdacc
第4個元素的位址：	0x7ffee46cdad0
第5個元素的位址：	0x7ffee46cdad4
第6個元素的位址：	0x7ffee46cdad8
第7個元素的位址：	0x7ffee46cdadc

陣列的指標

- 當一個陣列被宣告時，**它的名稱**可當作指標使用，指向該型態陣列的第一個元素。

```
array == &array[0]
```


將指標傳遞進函數

- 在定義函數雛型時，可將引數型態設定為**指標型態**。
- 在傳遞大陣列時，傳遞陣列指標比傳遞整個陣列來得快許多。
- 此種方式稱作「call-by-reference」。

函數雛型(Declaration)

用來告訴編譯器，這個程式會有哪些函數。

```
型態 函數名稱(引數1型態, 引數2型態, ...);
```

函數宣告(Definition)

用來定義一個函數實際的執行內容。

```
型態 函數名稱(引數1, 引數2, ...){  
    程式敘述;  
    ...  
}
```

練習

- (8-9) 寫一C++程式，將字串中的小寫轉成大寫。
 - 定義一個**toUpper**函數，接收呼叫敘述傳遞的字串指標參數，然後將字串中所有的小寫轉成大寫，其餘的字元不變，最後輸出轉換後的字串。
 - 在**main**函數中，定義一個字串指標，由鍵盤輸入一字串並存入指標位址，然後呼叫並傳遞字串給**toUpper**函數。
- (8-12) 寫一C++程式，將字串反向後回存並輸出。
 - 定義一個**reverse**函數，接收呼叫敘述傳遞的字串指標參數，然後將字串頭尾對調後返回呼叫函數。
 - 在**main**函數中，定義一個字串指標，由鍵盤輸入一字串並存入指標位址，然後呼叫並傳遞字串給**reverse**函數，最後輸出反向後的字串。

練習

- 變數交換程式（進階）
 - 當一個程式，時常需要交換兩個變數時，通常我們會自訂一個函數以完成這個功能，但之前所寫的函數只適用於int型態的變數。
 - 如果要交換很多種型態的變數，那就會對各種型態的變數都定義一個函數。
- 有沒有方法可以解決這個問題呢？

練習

- 變數交換程式（進階）

- 使用 **byte-by-byte** 的方式，將兩個變數中的值，一個個 **byte** 互相交換。
- 定義一個函數 `swap(void*, void*, int)`，輸入的三個參數分別是「第一個變數的指標」、「第二個變數的指標」、「變數的長度」。
- 特別注意事項：使用此函數時，會將指標強制轉型為 **void***，**void** 型態是沒有長度的。

記憶體空間								
	88	89	8A	8B	8C	8D	8E	8F
值	00001000	10100011	01001001	10100101	11011010	01001001	01101110	00000000
	↕	↕	↕	↕	↕	↕	↕	↕
值	10100010	10001100	00010100	10101010	01001101	01101101	01101011	11111010
記憶體空間	98	99	9A	9B	9C	9D	9E	9F

練習

- 變數交換程式（進階）
 - 使用 **byte-by-byte** 的方式，將兩個變數中的值，一個個 **byte** 互相交換。
 - 定義一個函數 `swap(void*, void*, int)`，輸入的三個參數分別是「第一個變數的指標」、「第二個變數的指標」、「變數的長度」。
 - 特別注意事項：使用此函數時，會將指標強制轉型為 **void***，**void** 型態是沒有長度的。
- 設計概念
 1. 要一次交換一個 **byte** 的話，可以將函數吃進來的指標強制轉型為 **char***。
 2. 因為我們不確定要交換的變數的型態的長度，因此要多一個參數來記錄型態長度。

本節概要

- 指標
 - 動態記憶體
 - 配置與釋放
 - 動態陣列
- C++字串類別：string
- 「製作OOXX遊戲」講解
- 「解數獨」講解

動態記憶體

- 程式不會自動回收不再使用的變數或陣列記憶體。
- 若程式需要使用很多變數或陣列，佔據的記憶體就會越來越多。
- 記憶體使用過量所產生的問題
 - 程式可用空間不足（現今一台電腦最多也差不多就**128GB**而已）
 - 程式執行的速度（影響存取變數的速度）
- 若有些變數或陣列不再使用，想要釋放佔用的記憶體空間，則可以使用**配置動態記憶體**的方式。

動態記憶體－變數

- new 運算符

- 用來配置動態記憶體，並傳回一個起始指標。
- 配置失敗時，回傳NULL值。

```
變數指標 = new 資料型態(起始資料);
```

- delete 運算符

- 用來釋放動態記憶體指標。
- 只能用來釋放已配置的動態記憶體指標。

```
delete 變數指標;
```


動態記憶體－變數

```
→ int *ptr  
ptr = new int;  
*ptr = 20;  
delete ptr;  
ptr = new int(30);  
delete ptr;  
  
ptr = new int(100);  
ptr = new int(80);  
delete ptr;
```

名稱：ptr
值：(未初始化)
型態：int*
長度：4 bytes
位址：0x0012FF9D

0x0012FF9D	0x0012FFA1	0x0012FFA5	0x0012FFA9
0x02AD21F0	0x02AD21F4	0x02AD21F8	0x02AD21FC
0x001224FA	0x001224FE	0x00122502	0x00122506

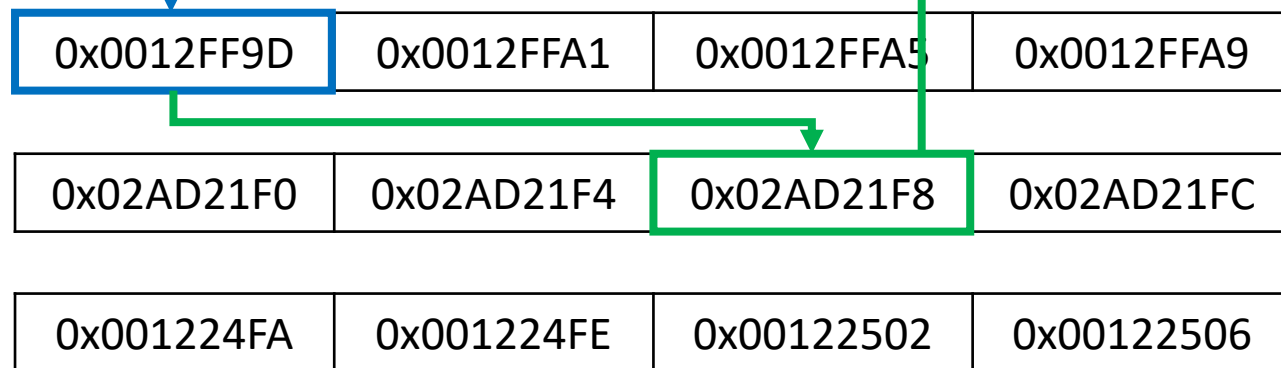
記憶體空間

動態記憶體－變數

```
int *ptr  
ptr = new int;  
*ptr = 20;  
delete ptr;  
ptr = new int(30);  
delete ptr;  
  
ptr = new int(100);  
ptr = new int(80);  
delete ptr;
```

名稱：ptr
值：0x02AD21F8
型態：int*
長度：4 bytes
位址：0x0012FF9D

名稱：(無)
值：(未初始化)
型態：int
長度：4 bytes
位址：0x02AD21F8



記憶體空間

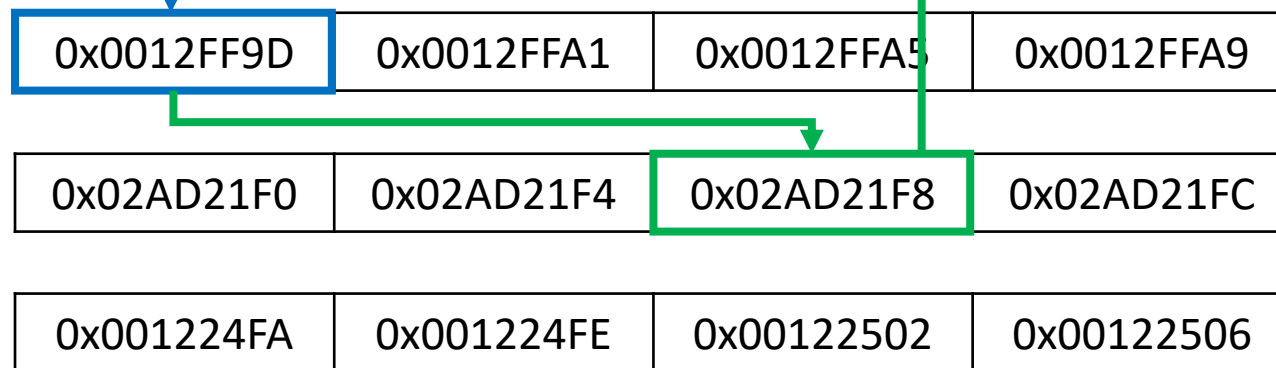
動態記憶體－變數

```
int *ptr
ptr = new int;
→ *ptr = 20;
delete ptr;
ptr = new int(30);
delete ptr;

ptr = new int(100);
ptr = new int(80);
delete ptr;
```

名稱：ptr
值：0x02AD21F8
型態：int*
長度：4 bytes
位址：0x0012FF9D

名稱：(無)
值：20
型態：int
長度：4 bytes
位址：0x02AD21F8



記憶體空間

動態記憶體－變數

```
int *ptr
ptr = new int;
*ptr = 20;
delete ptr;
ptr = new int(30);
delete ptr;

ptr = new int(100);
ptr = new int(80);
delete ptr;
```

名稱：ptr
值：NULL
型態：int*
長度：4 bytes
位址：0x0012FF9D

0x0012FF9D	0x0012FFA1	0x0012FFA5	0x0012FFA9
0x02AD21F0	0x02AD21F4	0x02AD21F8	0x02AD21FC
0x001224FA	0x001224FE	0x00122502	0x00122506

記憶體空間

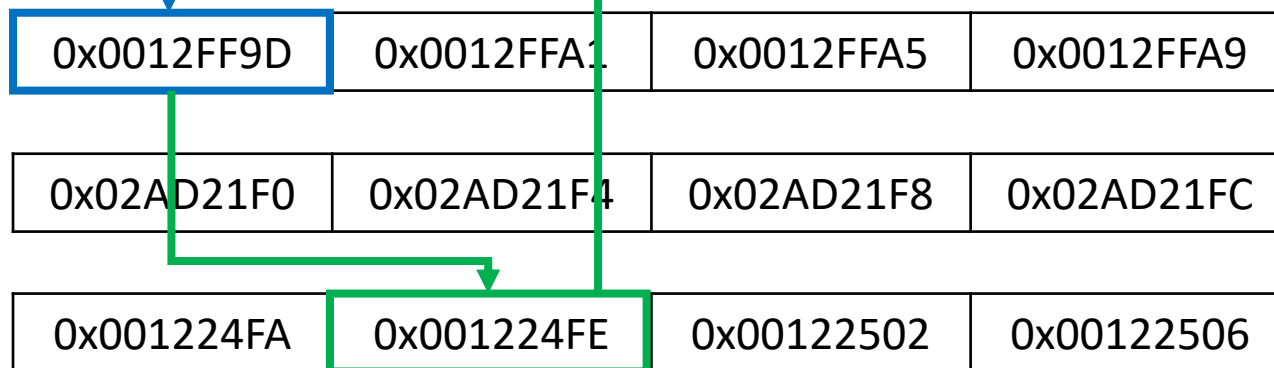
動態記憶體－變數

```
int *ptr
ptr = new int;
*ptr = 20;
delete ptr;
ptr = new int(30);
delete ptr;

ptr = new int(100);
ptr = new int(80);
delete ptr;
```

名稱：ptr
值：0x001224FE
型態：int*
長度：4 bytes
位址：0x0012FF9D

名稱：(無)
值：30
型態：int
長度：4 bytes
位址：0x001224FE



記憶體空間

動態記憶體－變數

```
int *ptr  
ptr = new int;  
*ptr = 20;  
delete ptr;  
ptr = new int(30);  
delete ptr;
```

```
ptr = new int(100);  
ptr = new int(80);  
delete ptr;
```

名稱：ptr
值：NULL
型態：int*
長度：4 bytes
位址：0x0012FF9D

0x0012FF9D	0x0012FFA1	0x0012FFA5	0x0012FFA9
0x02AD21F0	0x02AD21F4	0x02AD21F8	0x02AD21FC
0x001224FA	0x001224FE	0x00122502	0x00122506

記憶體空間

動態記憶體－變數

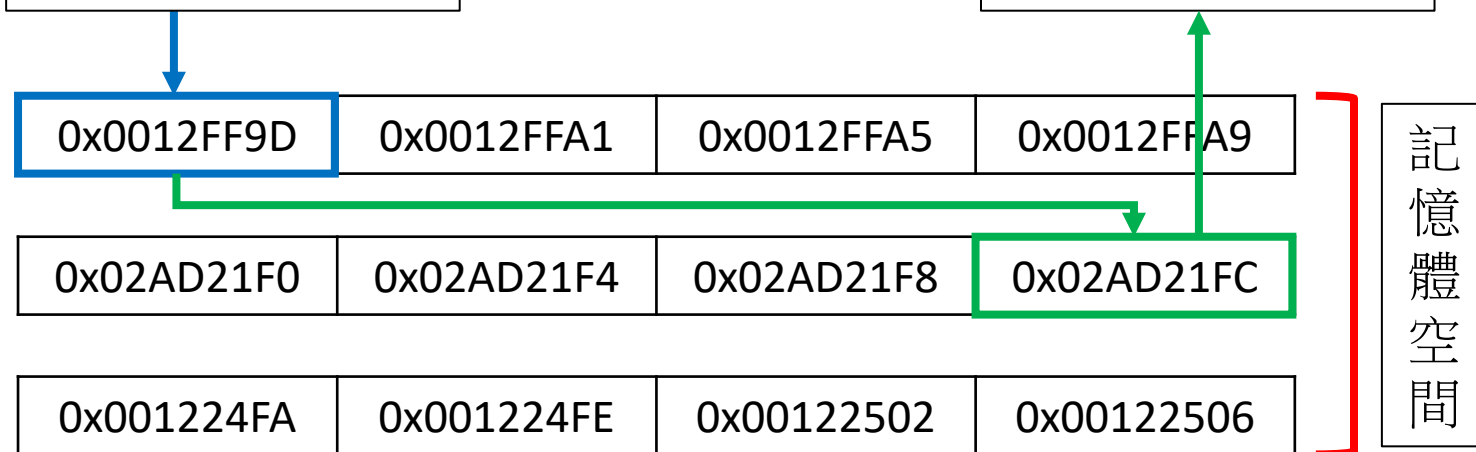
```
int *ptr  
ptr = new int;  
*ptr = 20;  
delete ptr;  
ptr = new int(30);  
delete ptr;
```

→

```
ptr = new int(100);  
ptr = new int(80);  
delete ptr;
```

名稱：ptr
值：0x02AD21FC
型態：int*
長度：4 bytes
位址：0x0012FF9D

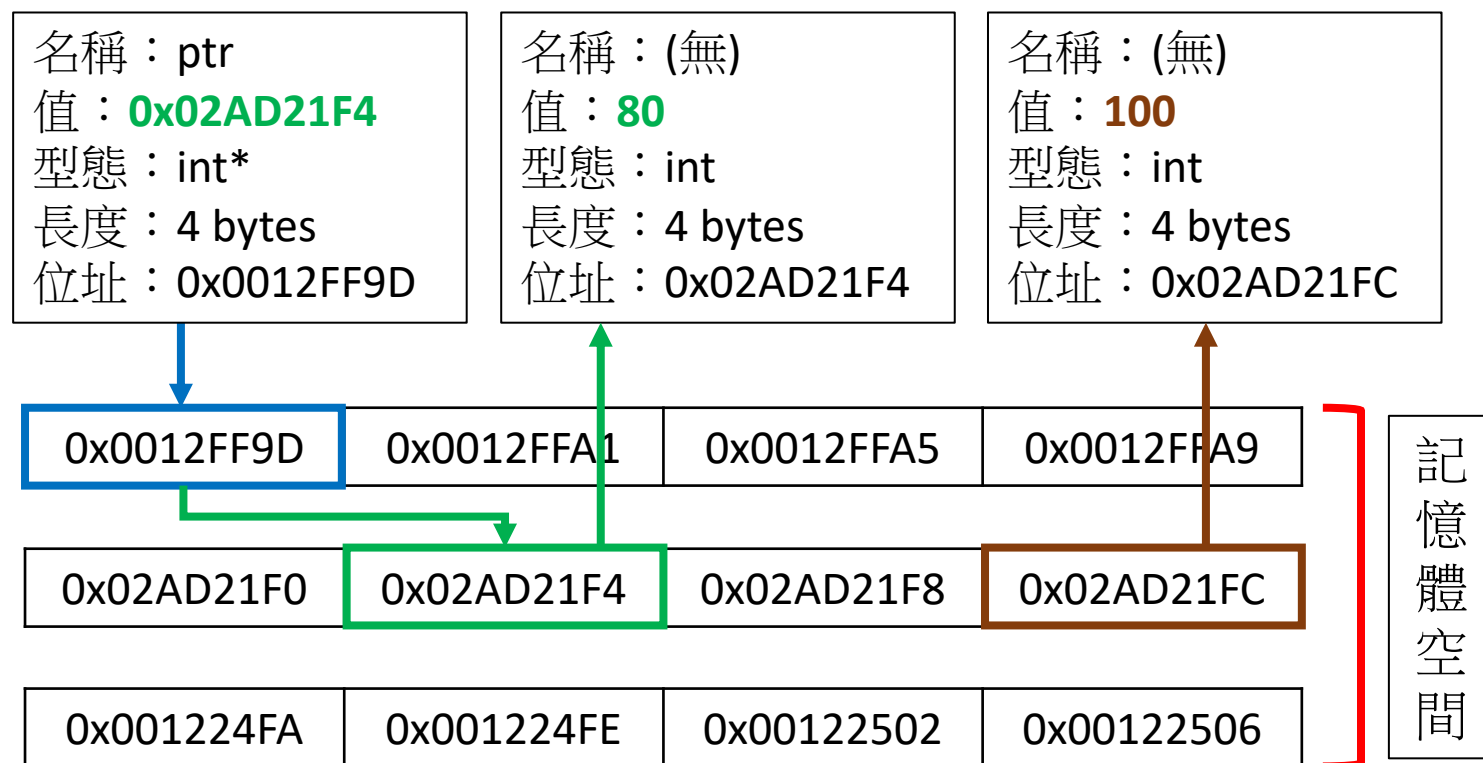
名稱：(無)
值：100
型態：int
長度：4 bytes
位址：0x02AD21FC



動態記憶體－變數

```
int *ptr
ptr = new int;
*ptr = 20;
delete ptr;
ptr = new int(30);
delete ptr;

ptr = new int(100);
ptr = new int(80);
delete ptr;
```



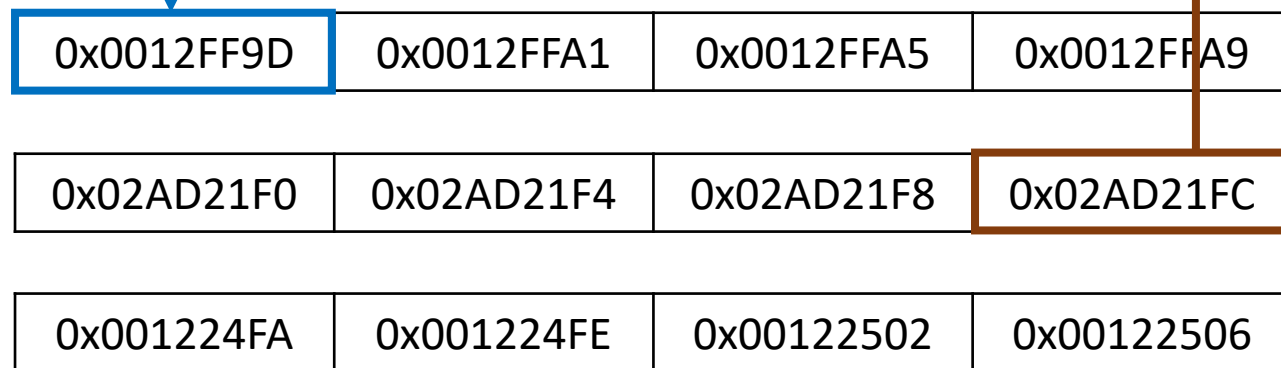
動態記憶體－變數

```
int *ptr
ptr = new int;
*ptr = 20;
delete ptr;
ptr = new int(30);
delete ptr;

ptr = new int(100);
ptr = new int(80);
delete ptr;
```

名稱：ptr
值：NULL
型態：int*
長度：4 bytes
位址：0x0012FF9D

名稱：(無)
值：**100**
型態：int
長度：4 bytes
位址：0x02AD21FC



記憶體空間

Memory leak (記憶體流失)

- 若在用**delete**釋放記憶體前，就將指標指向其它地方，會發生什麼問題？
- 記憶體流失
 - 可以使用的記憶體越來越少。
 - 若記憶體用光，會導致程式無法運作。
- 參考說明：
<https://zh.wikipedia.org/wiki/%E5%86%85%E5%AD%98%E6%B3%84%E6%BC%8F>

練習

- 寫一個C++程式，練習動態記憶體配置與釋放。
 1. 宣告3個int指標，並動態配置記憶體，初始值分別為400, 200, 100。
 2. 印出3個int指標指向的記憶體位址與值。
 3. 宣告2個float指標，並動態配置記憶體，初始值分別為3.14, 6.28。
 4. 印出2個float指標指向的記憶體位址與值。
 5. 釋放3個int指標。
 6. 重新配置動態記憶體給3個int指標，初始值分別為10, 50, 100。
 7. 印出3個int指標指向的記憶體位址與值。
 8. 釋放2個float指標。
 9. 釋放3個int指標。

動態記憶體－陣列

- new 運算符
 - 與配置動態變數指標類似。
 - 「new 資料型態」後面，要加上「[長度]」，用來配置陣列長度。

陣列指標 = new 資料型態[長度]

- delete 運算符
 - 與釋放動態變數指標類似。
 - delete 運算符之後，必須加上中括號，表示被釋放的指標是陣列指標。
 - 只能用來釋放已配置的動態記憶體指標。

delete[] 陣列指標;

「製作OOXX遊戲」講解

「解數獨」講解

程式設計思路

- 對於每一個空格，分別確認

- 同一個九宮格

- 同一橫列

- 同一直行

● = 1, 2, 3, 4, 5, 6, 7, 8, 9

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

程式設計思路

- 對於每一個空格，分別確認
 - 同一個九宮格
 - 同一橫列
 - 同一直行

● = 1, 2, 3, 4, 5, 6, 7, 8, 9

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

程式設計思路

- 對於每一個空格，分別確認
 - 同一個九宮格
 - 同一橫列
 - 同一直行

● = 1, 2, 3, 4, 5, 6, 7, 8, 9

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

程式設計思路


- 對於每一個空格，分別確認

- 同一個九宮格

- 同一橫列

- 同一直行

 = 1, 2, 3, 4, 5, 6, 7, 8, 9

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

下週預計課程內容

- STL container
 - vector
 - map

延伸閱讀

- C/C++ - 常見 C 語言觀念題目總整理（適合考試和面試）
 - <http://mropengate.blogspot.com/2017/08/cc-c.html>

作業

- 從以下題目任選兩題完成，下次上課時找助教檢查。
 - d881：作業苦多
 - d122：Oh! My Zero!!
 - d086：態度之重要的證明
 - d566：秒殺率
 - b701：我的領土有多大
 - b523：先別管這個了，你聽過安麗嗎？
 - d527：程式設計師的面試問題(三)
- Reading: 課本Ch8.4, 9.4
- 若遇到作業問題，歡迎隨時寄信至：ck1001099@gmail.com