C++程式設計基礎
week 6

陳毅

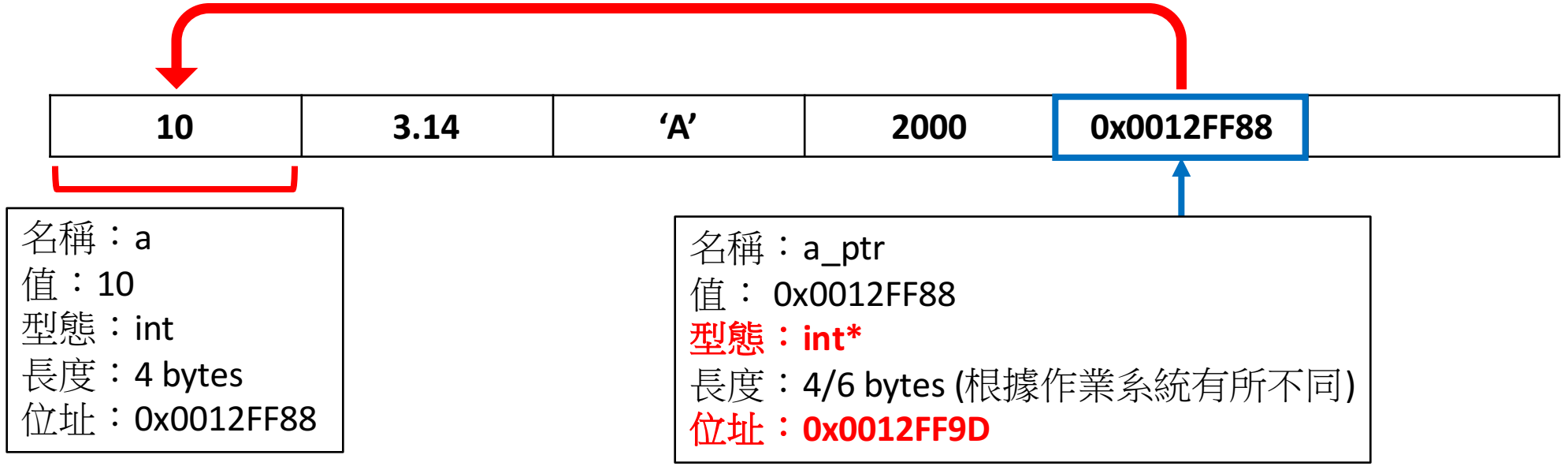
上週回顧

- 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
```

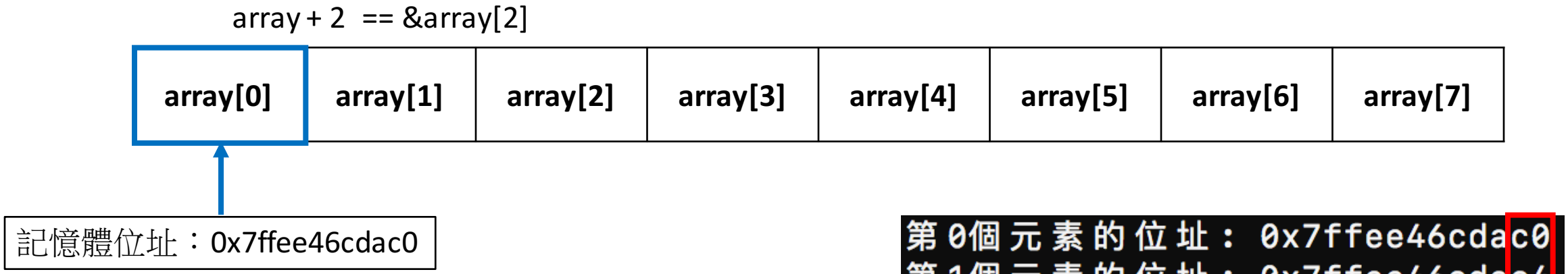
- 取得一個記憶體位址所儲存的**值**
  - 「*」稱作**間接運算符號(indirect operator)**，是用來取得參考位址內的值，也稱作**反參考運算符號(de-reference operator)**。

```
*指標名稱
```

5-1.cpp, 5-2.cpp

陣列的指標

- 一維陣列（以int[8]為例）



array[i] = array[0] + i * sizeof(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。

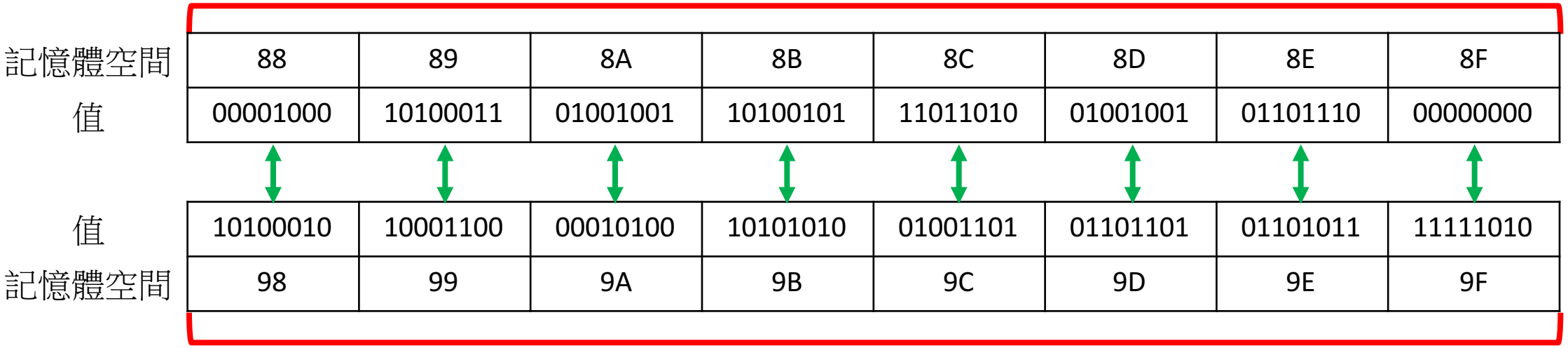
練習

- 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是沒有長度的。**



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變數是沒有長度的。**

  - 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。

本週概要

- 指標
  - **動態記憶體**
    - **配置與釋放**
    - **動態陣列**
- C++字串類別：string
- 「期末專案：踩地雷」介紹

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。

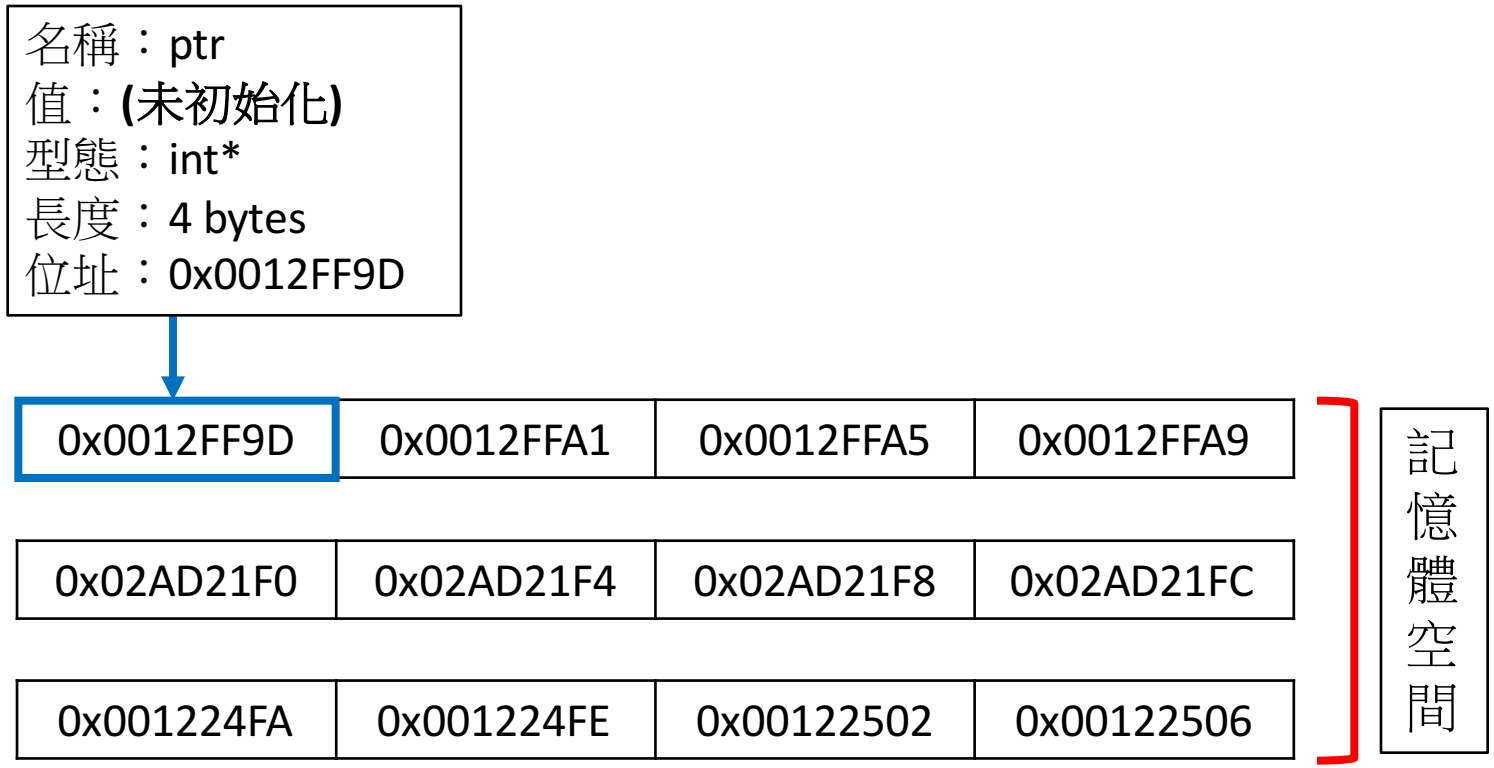
```
變數指標 = new 資料型態(起始資料);
```

- 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

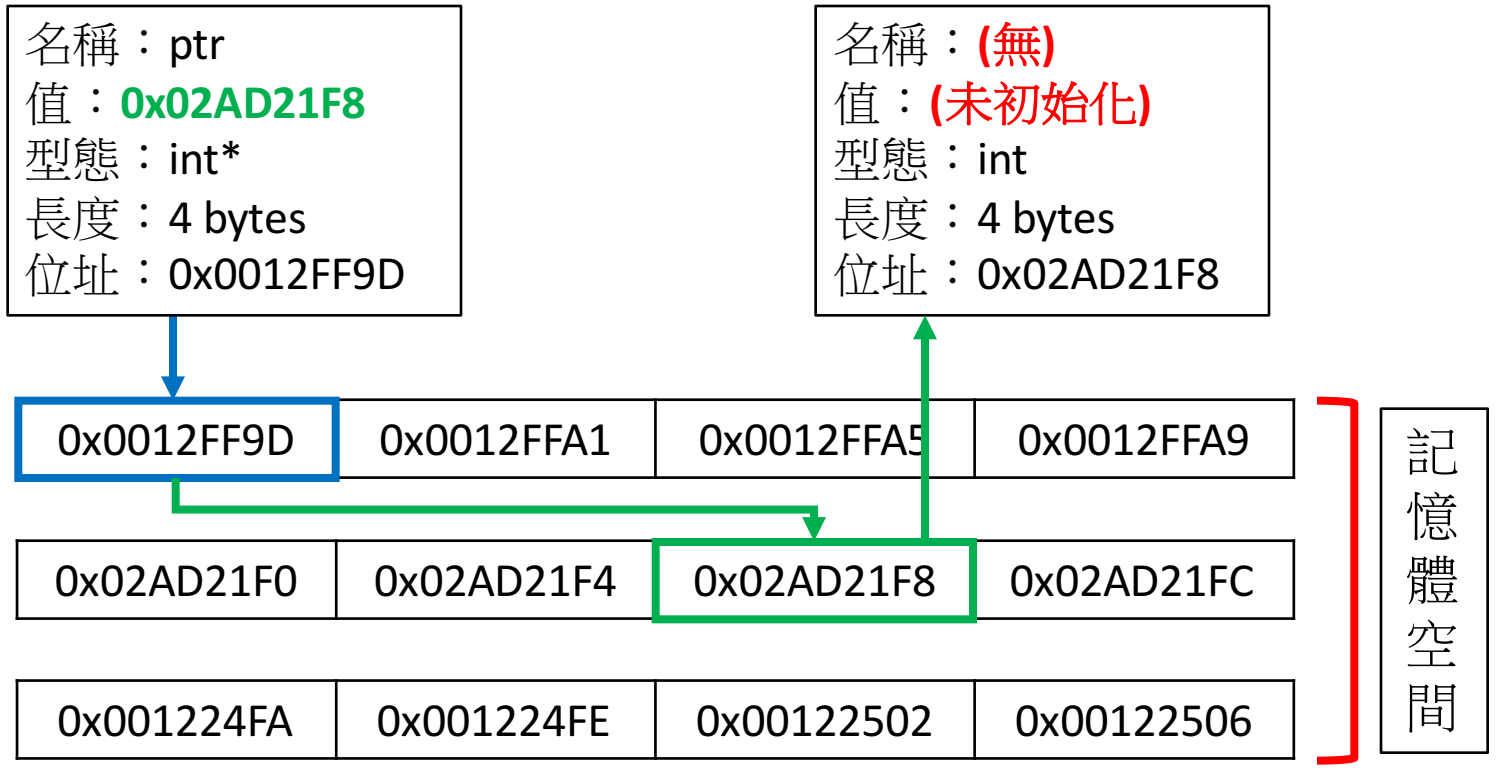
```
int *ptr
ptr = new int;
*ptr = 20;
delete ptr;
ptr = new int(30);
delete ptr;

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



6-1.cpp

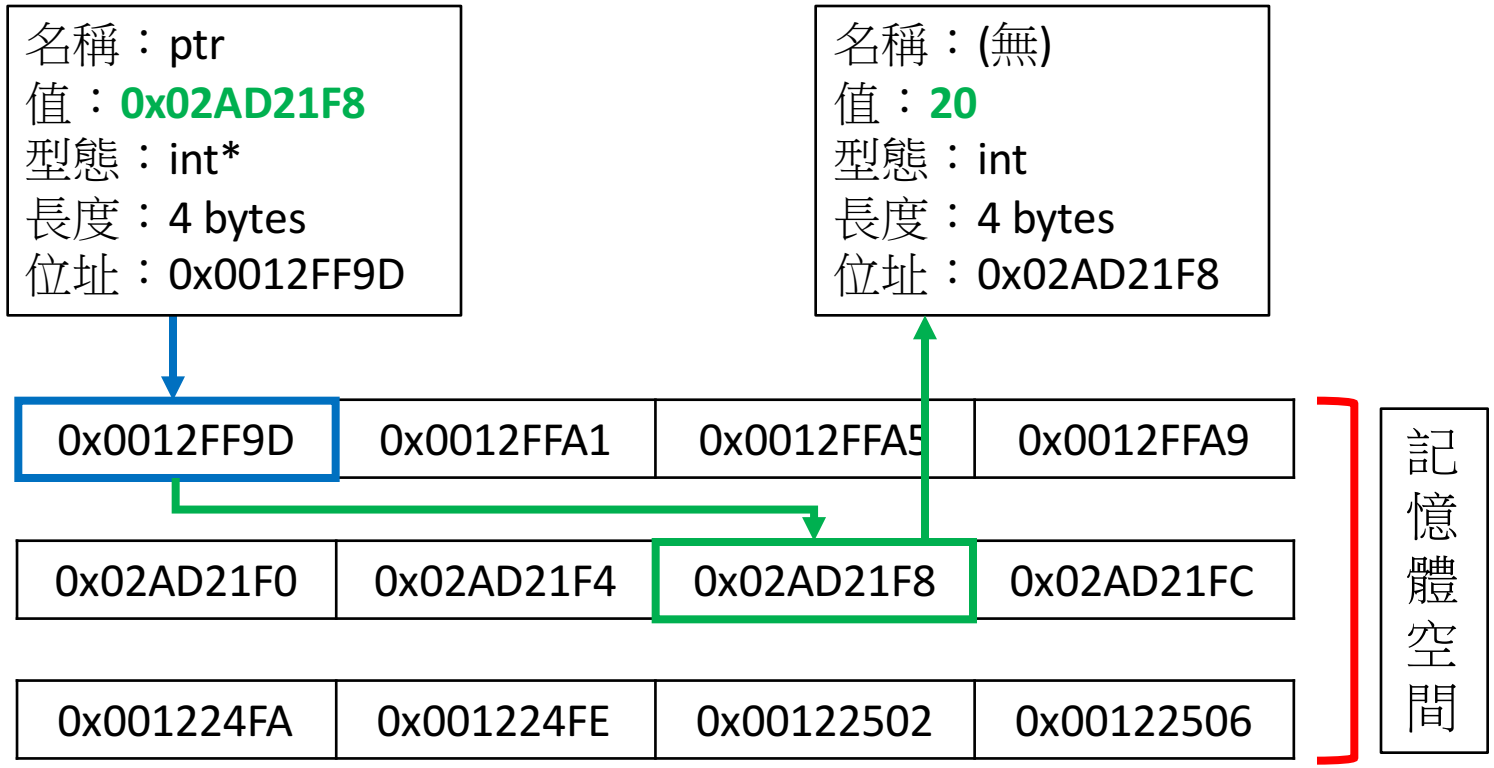
動態記憶體 – 變數

```
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;
```



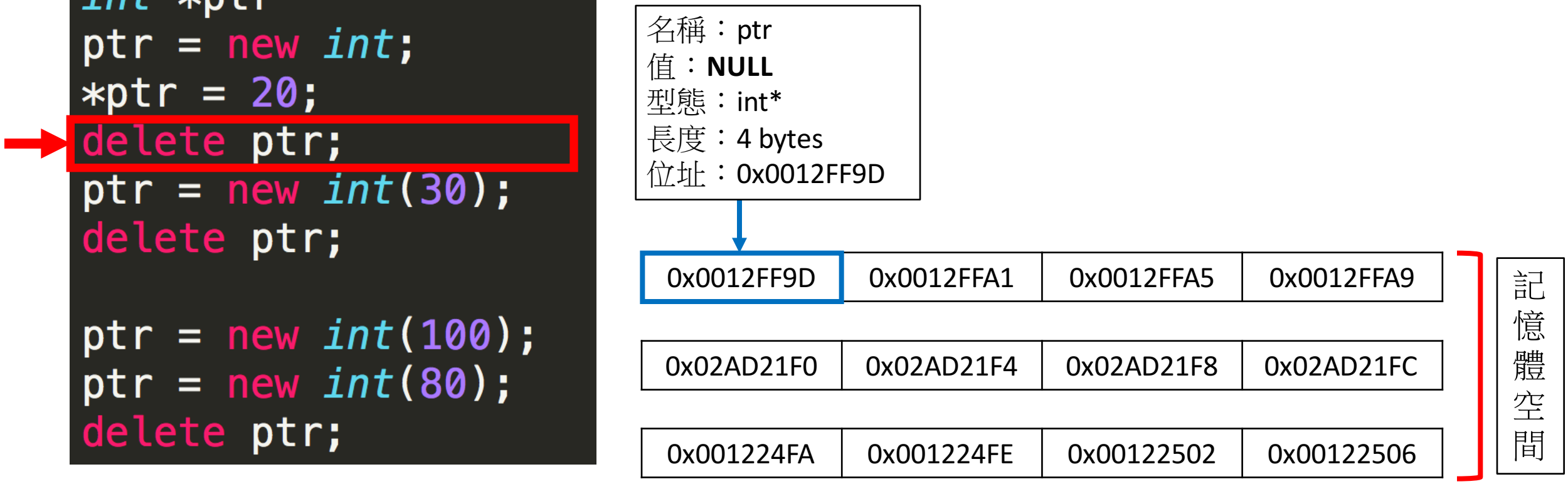
動態記憶體 – 變數

```
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;
```



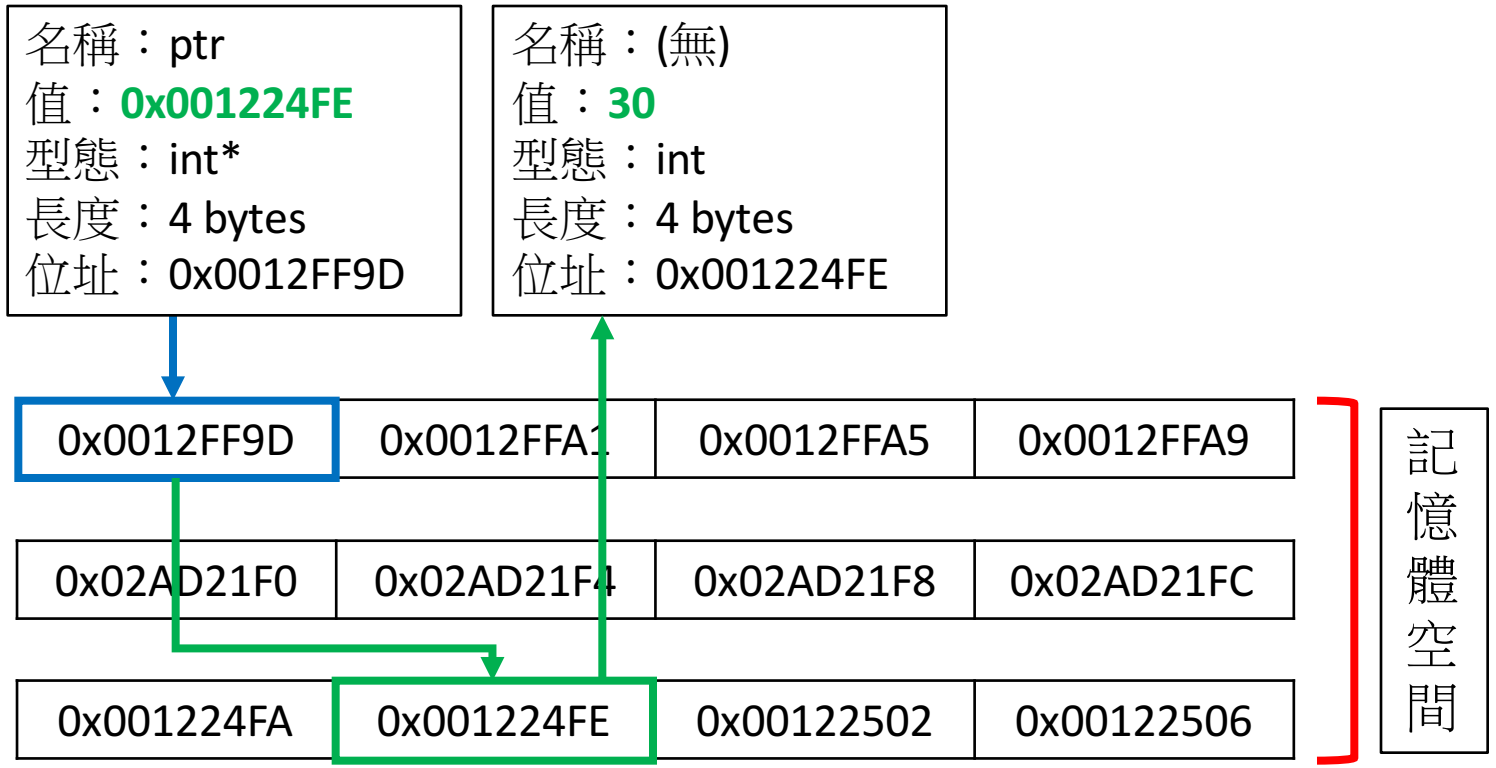
動態記憶體 – 變數

```
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;
```



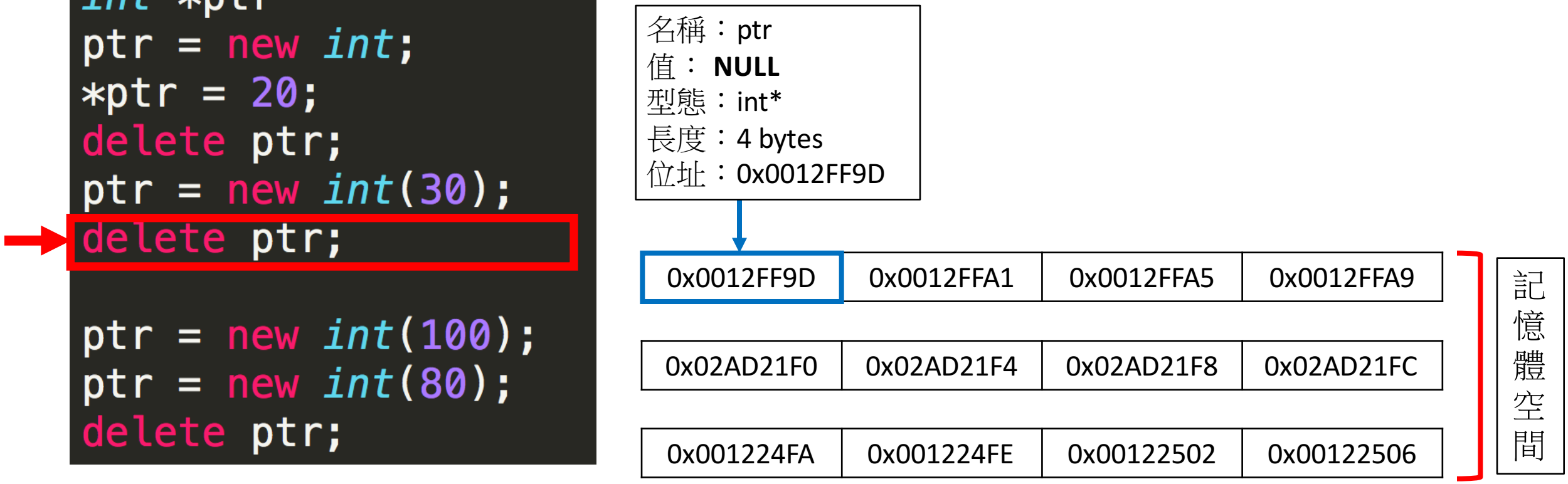
動態記憶體 – 變數

```
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;
```



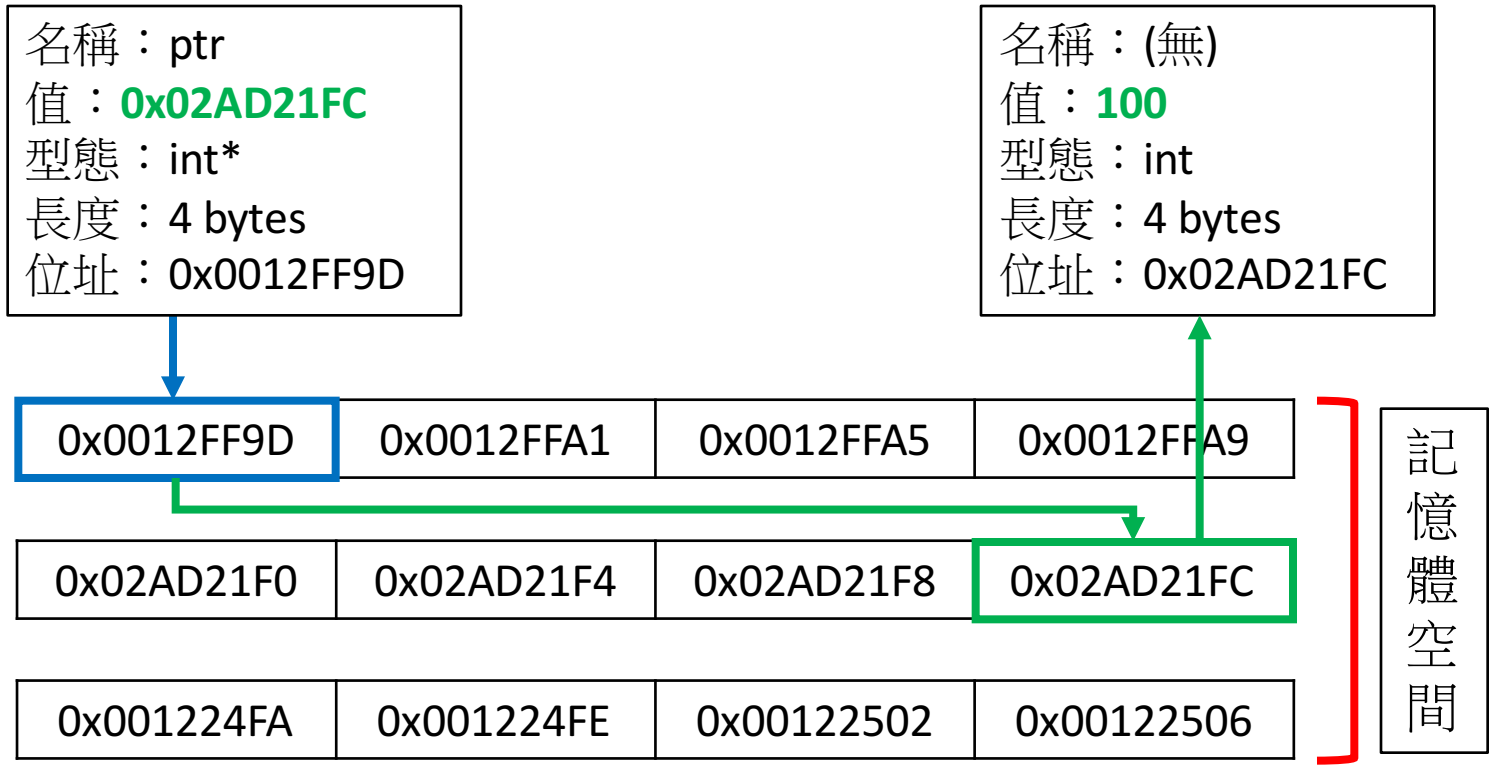
動態記憶體 – 變數

```
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;
```



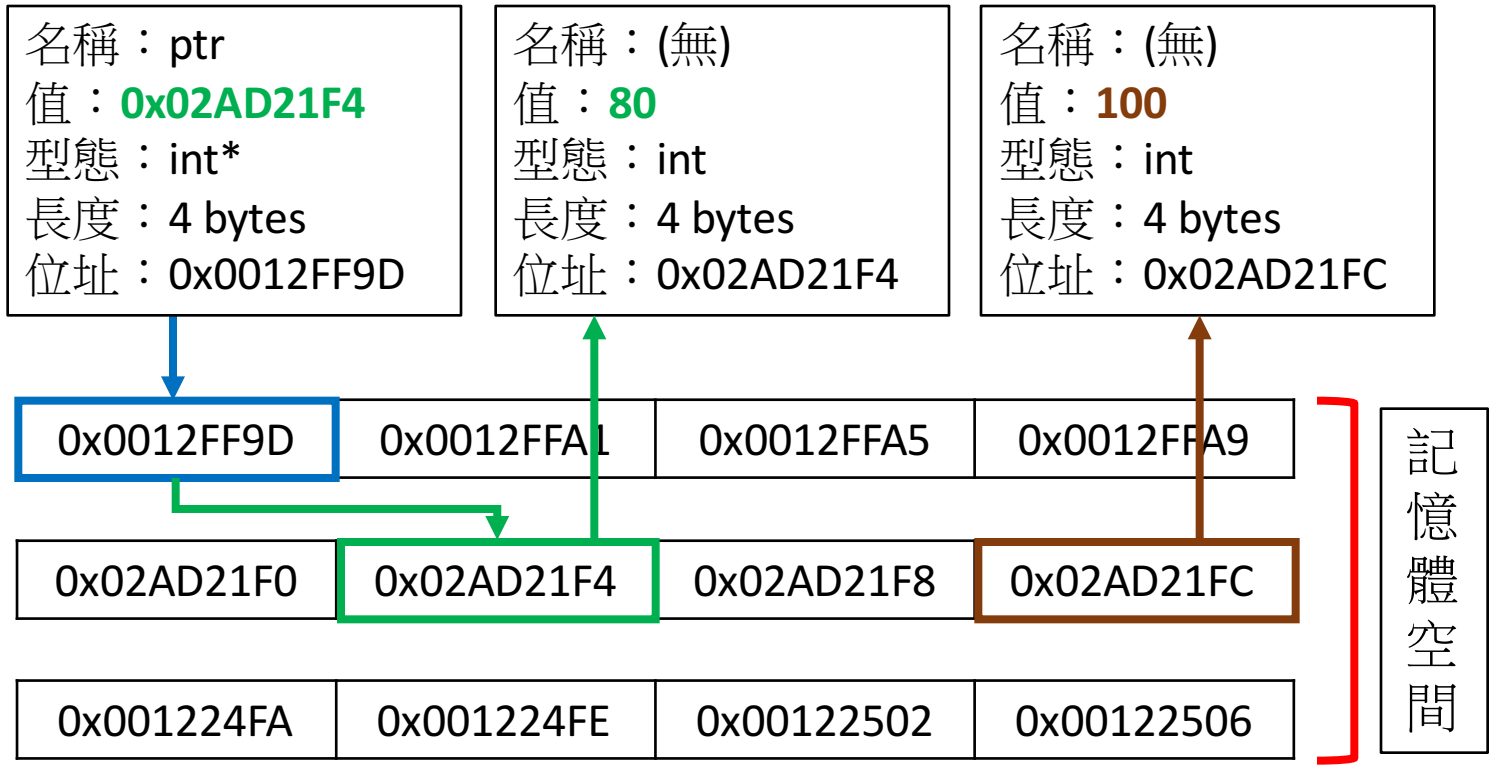
動態記憶體 – 變數

```
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;
```



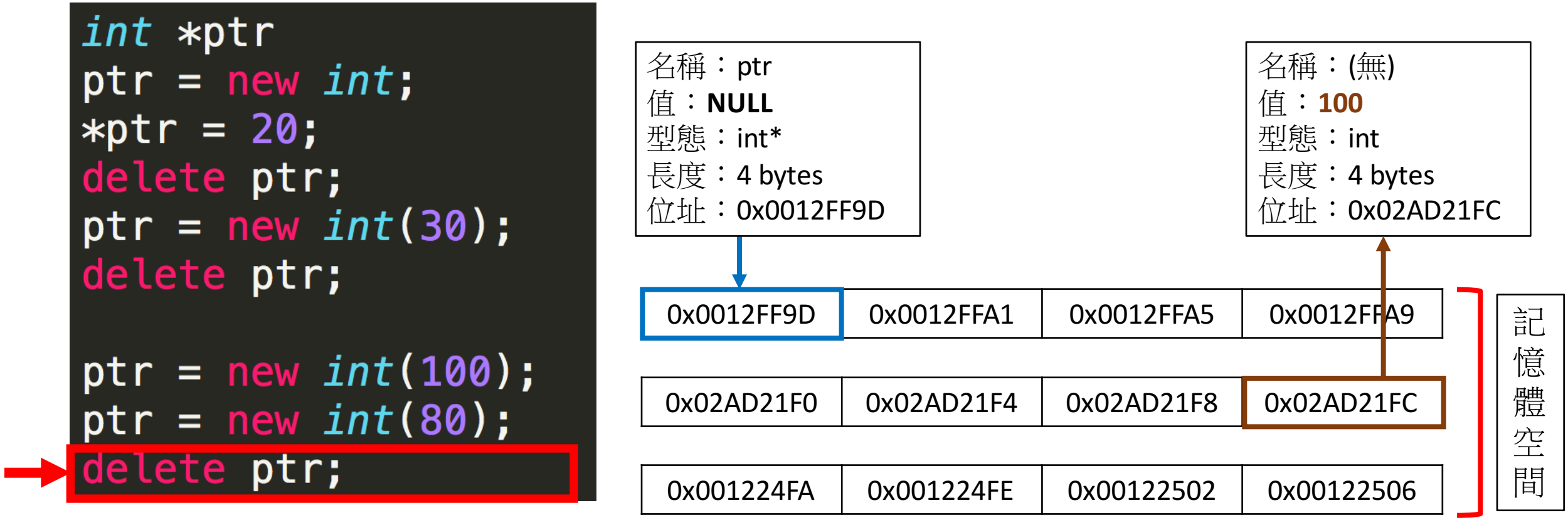
動態記憶體 – 變數

```
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;
```



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

6-2.cpp

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;
```

6-3.cpp, 6-4.cpp

C++字串類別：string

簡介

- C++字串類別是一個抽象的資料型態，它並**不是C++原本內建的資料型態**。
- C++字串類別及其相關函數式定義於C++的新型標題檔(header)中，因此使用這些函數前，必須將其引入。

- 要引入標頭檔：#include <string>
- 被定義在std下，因此要輸入using namespace std;

C型態字串 v.s. C++字串類別

- **C型態字串**：使用字元陣列或指標來定義字串。

```
char *name = "JOHN";
char name[20] = "JOHN";
```

- **C++字串類別**：宣告string類別的字串物件來處理字串。

```
string s1;
string s2("JOHN ARCHER");
string s3 = "MARY ARCHER";
string s4("A", 4);
string s5(s2);
string s6(s2, 0, 4);
```

C++字串類別

- 宣告及初始化方式
  - C++ reference: http://www.cplusplus.com/reference/string/string/string/
- 輸入C++字串
  - cin >> 字串物件;
  - getline(cin, 字串物件);

```
string s1, s2;

cout << "請輸入 s1 字串: ";
getline(cin, s1);
cout << "請輸入 s2 字串: ";
cin >> s2;
cout << "s1 = " << s1 << endl;
cout << "s2 = " << s2 << endl;
```

6-5.cpp

C++字串類別

- 常用運算符號

運算符號	功能說明
=	指定資料
+	串接字串
+=	連接並指定字串
==	相等
!=	不相等
<, >, <=, >=	逐一比較字元大小
[ ]	存取字元
<<	輸出
>>	輸入

6-6.cpp, 6-7.cpp

C++字串類別

- 字串物件陣列：與宣告一般陣列無異。

```
string array[10];
```

```
string s1[] {"Java", "Assembly", "Delphi", "Basic", "Fortran", "Cobol"};
```

6-8.cpp

C++字串類別

- C++字串類別成員函數（參見課本**表9.2**）
  - C++ reference: http://www.cplusplus.com/reference/string/string/

成員函數	功能
s1.append(s2)	連接字串
s1.at(位置)	存取指定位置
s1.clear()	清除字串全部內容
s1.length()	取得字串長度
s1.swap(s2)	對調字串
s1.replace(起始位置, 字串長度, s2)	取代部分字串
s1.insert(起始位置, s2)	插入字串
s1.find(s2)　s1.find(s2,起始位置)	找尋字串
s1.copy(s2, 起始位置, 字串長度)	複製字串

練習

- (9-13) 寫一C++英打練習程式。
  - 在main函數中，定義**一個字串陣列**，起始資料為英文單字（最少30個單字），再定義**一個字串變數**用來存放串接後的字串。
  - 利用亂數取得陣列中的5個單字，將5個單字串接在一起，單字間以空白隔開。再將此字串存入字串變數後，呼叫並傳遞字串參數給englishTyping函數。
  - 定義一個englishTyping函數，接收呼叫敘述傳遞的字串參數，然後接收鍵盤輸入，按Enter鍵後輸出正確字數、錯誤字數、與正確率。

期末專案：踩地雷

說明

- 維基百科：https://zh.wikipedia.org/wiki/%E8%B8%A9%E5%9C%B0%E9%9B%B7

- 基礎規則
  - 踩到地雷就Game over。
  - 如果不是地雷的格子，但與地雷相連，會顯示自己周圍有幾個地雷。
  - 找出全部的地雷就獲勝

說明

- 建立在基礎規則上自由發揮（必要時，也可改變基礎規則）
- 例如
  1. 改變遊戲盤面，不要做成正方形的。
  2. 玩家選取的第一個點，絕對不會是地雷
  3. 新增「讓玩家自由創作盤面」的功能
  4. 有三條命
  5. 雙人PK找地雷模式
  6. 時間限制

說明

- 以小組為單位，一組1至4人，找好請找助教登記。

- 11/14：上課 + 課堂實作
- 11/21：上課 + 課堂實作
- 11/28：上課 + 課堂實作
- 12/3前：上傳「期末專案」程式碼及投影片
  - 程式碼：.cpp檔、.h檔 等。
  - 投影片：簡單的投影片即可，用於向大家介紹專案內容。
  - 下次上課公布上傳地點。
- 12/5：Demo

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：r07922059@ntu.edu.tw