Chapter 6
Pointers

Yu-Hsiang Cheng
鄭宇翔 (Slighten)
GOTC Instructor
http://m101.nthu.edu.tw/~s101021120/Myweb/index_ch.html

Outline

1. 指標介紹
2. 指標與陣列之關係
3. 剩下 OOP 以前內容的介紹

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

What is a Pointer (指標)?

a memory address (記憶體位置)

- an `unsigned int` (非負整數) – 4 bytes

但這個詞有點模糊，它可以指

1. 一個指標變數(一個存記憶體位置的變數)
2. 一個 address expression 的值 (一個記憶體位置)

一個指標允許間接存取

- To reference (參考) a memory location (via its address)
- To dereference (解參考、參照) a pointer (to get to the content)

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

L-Value vs. R-Value of an Expression

L-Value ("left-hand-side" of an assignment)

- 有值也有記憶體位置
- 例：{int x; x = 3; y = x + 5;}

  x 有 L-value，因為 x 有一個記憶體位置並且可以存值
- L-value 在算完 expression 後仍然存在.

R-Value

- 值的本身，與記憶體位置無關

  ⇒不能在等號左邊 (不能被指派)
- 例：{int x, y; x = 3;}

  3 只有 R-value. 而且，不能做 3 = x;
- 例： y = (x + 5);

  x 有 R-value of 3

  (x + 5)有 R-value of 3 + 5 = 8.

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

More about L-Value vs. R-Value

A variable

- 有 L-value 和 R-value, 看它是在等號左邊或右邊

An int, char, floating-point literal (`2, 3.1415, 'K'`)

- 只有 R-value, 沒有 L-value

A string constant (`"hello"`)

- 有 L-value 和 R-value，但不能被更改(不能放等號左邊)(OS-限制, immutable)
- 存在字串常量池裡

An indexed expression (`A[i]`, `A` 是個 array)

- 像變數，有 L-value 和 R-value

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

Declaration and Operations

Syntax

`int *p;` 或 `int* p;`

```
/* p is a pointer variable to an int */
```

也可以是指向其他型別的指標變數

`int*`, `double*`, `char*`, etc.

也可以是指向指標變數的指標變數

```
int **q; /* q is a pointer to a pointer to an int */
```

(unary) Dereference operator (解參考、參照運算子): *

- `*p` -- the content at the address expression p.
- It has both an L-value and an R-value, like a variable

•(unary) Address-of (reference) operator (位址運算子): &

- `&q` -- the address of variable q.
- It has only an R-value: CANNOT say `&q = 10;`

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

That's Why We CANNOT Assign an Array to Another Array

- `char str1[10] = "hello", str2[10];`
- `str2 = str1; // illegal`

- 因為 `str2 == &str2[0]`，而 `&str2[0]` 只有 R-value

⇒CANNOT say `&str2[0] = str1;`

⇒CANNOT say `str2 = str1;`

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

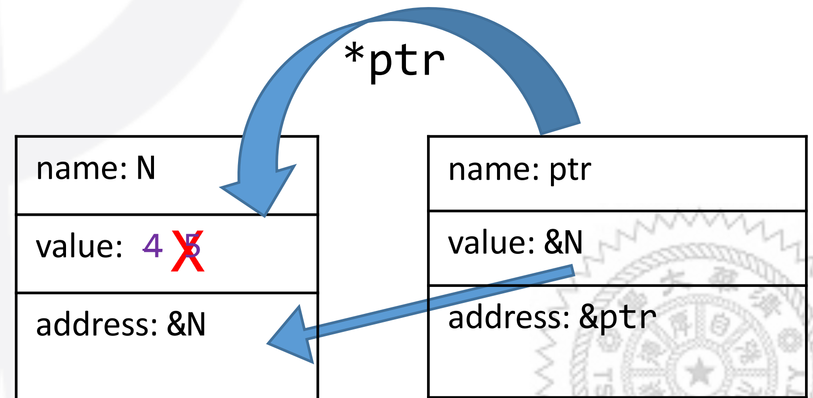
Example of pointer operators

```
int N;
int *ptr; /* declares ptr as a pointer variable to an int */
N = 4; /* store value 4 into variable N */
ptr = &N; /* store address of variable N into ptr */
*ptr = 5; /* store 5 at the address pointed to by ptr */
cout << "value of N = " << N << '\n';
```

=> this prints

N = 5

因為 N 的值透過 ptr 被間接修改了



Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

圖示 (傳值)

```
char b = 'c';
char a = b;
```

名稱：b

'c'

記憶體位置：&b == 13

'c'

'c'

名稱：a

記憶體位置：&a == 18

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

圖示 (傳參考)

```
char b = 'c';
char &a = b;
```

名稱：b

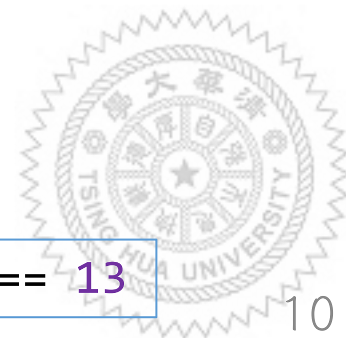
'c'

記憶體位置：&b == 13

名稱：b 或 a

'c'

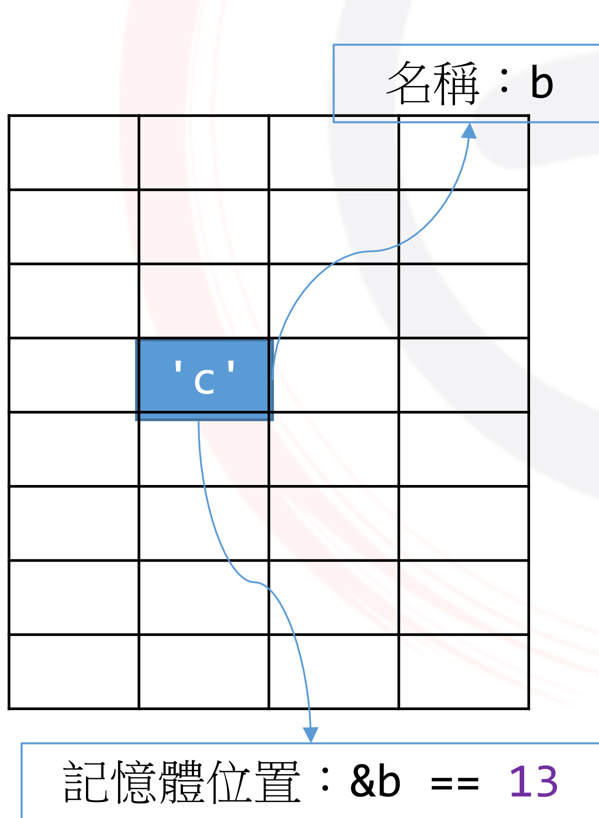
記憶體位置：&b 或 &a == 13

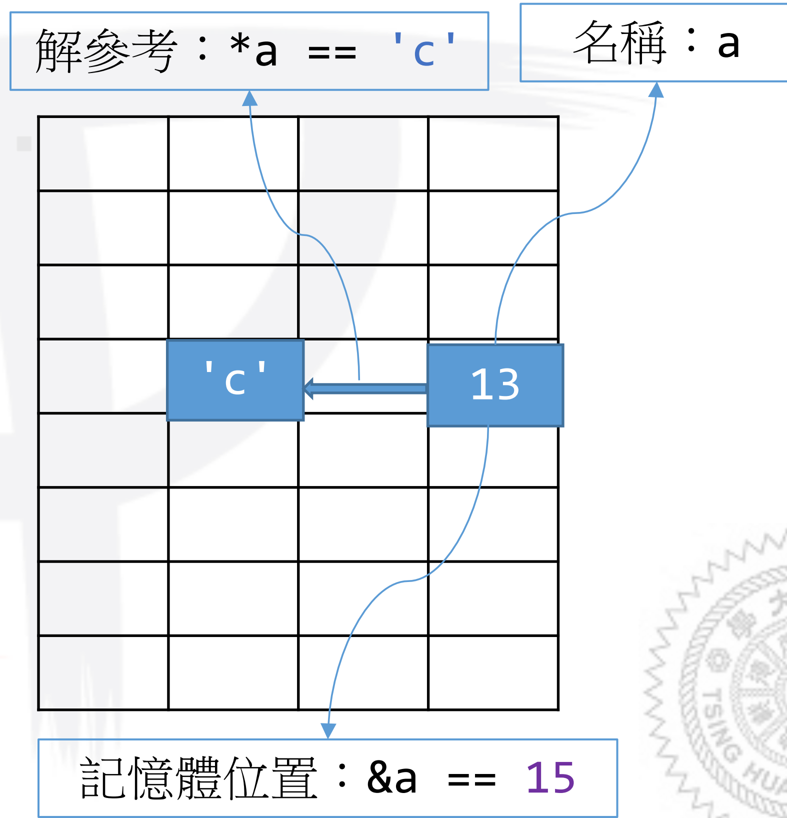


Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

圖示 (傳指標)

```
char b = 'c';
char *a = &b;
```





Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

Let’s Watch a Video

- Vsauce 刪掉的東西去哪了？ https://www.youtube.com/watch?v=ylx4vsWKOQI

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

Simple Pointer Example

```
void increment(int *n){
    (*n)++;
}

int main(){
    for (int i = 0; i < 10; increment(&i))
        cout << i << ' ';
    cout << endl;
}
```

- increment(&i) 就像做 i++
- (*n) 就是 main() 裡的 i

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

Need for Addresses

- Scalars (純量) (`int`, `char`, `float`...) are passed by value (傳值)

⇒ can't swap

```
void uselessSwap(int firstVal, int secondVal){
    int tempVal = firstVal;
    firstVal = secondVal;
    secondVal = tempVal;
}
int main() {
    int a = 3, b = 5;
    uselessSwap(a, b);
    cout << "a = " << a << ", b = " << b << endl;
}
```

- 是 firstVal 與 secondVal 在交換，而不是 a, b 在交換！
- 怎麼辦？ 方法有二

1. 傳 a, b 的「記憶體位置」！(傳指標)
2. 讓 firstVal, secondVal 成為 a, b 的「參考」(別名) (傳參考、只有 C++ 有的特色)

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

法一：傳指標 (較繁瑣)

```
void uselessSwap(int *firstVal, int *secondVal){
     int tempVal = *firstVal;
     *firstVal = *secondVal;
     *secondVal = tempVal;
}
int main() {
     int a = 3, b = 5;
     uselessSwap(&a, &b);
     cout << "a = " << a << ", b = " << b << endl;
}
```

- OK! 我們要把變數 a, b 的記憶體位置當參數傳給 uselessSwap()
- 問題有三

1. 如何取得 a, b 的記憶體位置？

   ⇒ 利用 & 運算子：&a, &b 就是 a, b 的記憶體位置

2. 哪種參數(變數)能存記憶體位置？

   ⇒ 利用「指標變數」來存記憶體位置：int *firstVal, *secondVal;

   ⇒ firstVal = &a; secondVal = &b;

3. 指標變數記了記憶體位置，如何存取那個記憶體位置？

   ⇒ 對指標變數利用 * 運算子去存取它存的記憶體位置的值：*firstVal, *secondVal;

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

法二：傳參考 (較簡單)

```
void uselessSwap(int &firstVal, int &secondVal){
     int tempVal = firstVal;
     firstVal = secondVal;
     secondVal = tempVal;
}
int main() {
     int a = 3, b = 5;
     uselessSwap(a, b);
     cout << "a = " << a << ", b = " << b << endl;
}
```

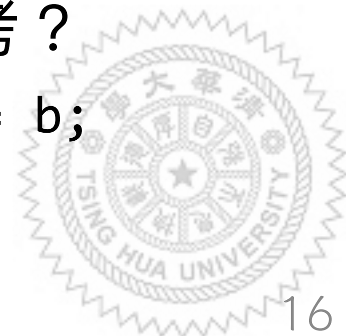
- 另一種方法是不傳記憶體位置，直接讓 firstVal, secondVal 成為 a, b 的參考 (匿名、別名、小名、化身) 就好，也就是讓 firstVal, secondVal 成為 a, b 「本身」！
- 問題只有一個

1. 如何讓 firstVal, secondVal 成為 a, b 的參考？

⇒ 利用「參考變數」：int &firstVal = a, &secondVal = b;

⇒ firstVal 就是 a！ secondVal 就是 b！

⇒ CSW就是張舜為！



Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

Null Pointer (空指標)

- 有時候我們希望指標變數先不要指任何東西(不要記任何記憶體位置)
- 這時就需要讓它指到「空指標」

```
int *p;
p = NULL; /* p is a null pointer */
```

- NULL 是一個 pre-defined macro，它有一個 non-null pointer 不會有的值
  - 通常， NULL == 0，因為位址 0 對大部分作業系統來說是一個非法的記憶體位置

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

Using Arguments for Results

- 想要讓 function 改的變數就傳記憶體位置
  - 這使一個 function 擁有能回傳多重結果的功能
  - 因為原本靠 return 只能回傳 0 或 1 個結果
- 事實上
- 在 C 語言裡 scanf() (in <cstdio>) 就像 C++ 的 cin
  - int num;
- C++: cin >> num;
- C: scanf("%d", &num);
- 在 C 語言裡 printf() (in <cstdio>) 就像 C++ 的 cout
- C++: cout << num << "\n";
- C: printf("%d\n", num);

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

想一想

- 為什麼 scanf() 需要變數的記憶體位置， cin 不用？
- 答： scanf() 傳指標， cin 的 >> 運算子(函式)傳參考！
- 事實上運算子 >> 的定義有點類似以下

```
istream& operator>>(some_type &n){
    n = extract(*this); // this points to cin
    return *this; // enables successive >>
}
```

- cin >> num1 >> num2;

1. 把 num1 當參數傳給 n
2. n 成為 num1 的匿名
3. n 拿到 cin 存的值
4. 回傳 cin，使可以做 cin >> num2;
5. 重複第一步將變數改成 num2

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

Pointer–Variable Declaration

- 宣告指標變數：
- `type *var;`
- `type* var;`
- 以上兩種都可，空白可忽略，`int*`, `char*`, …

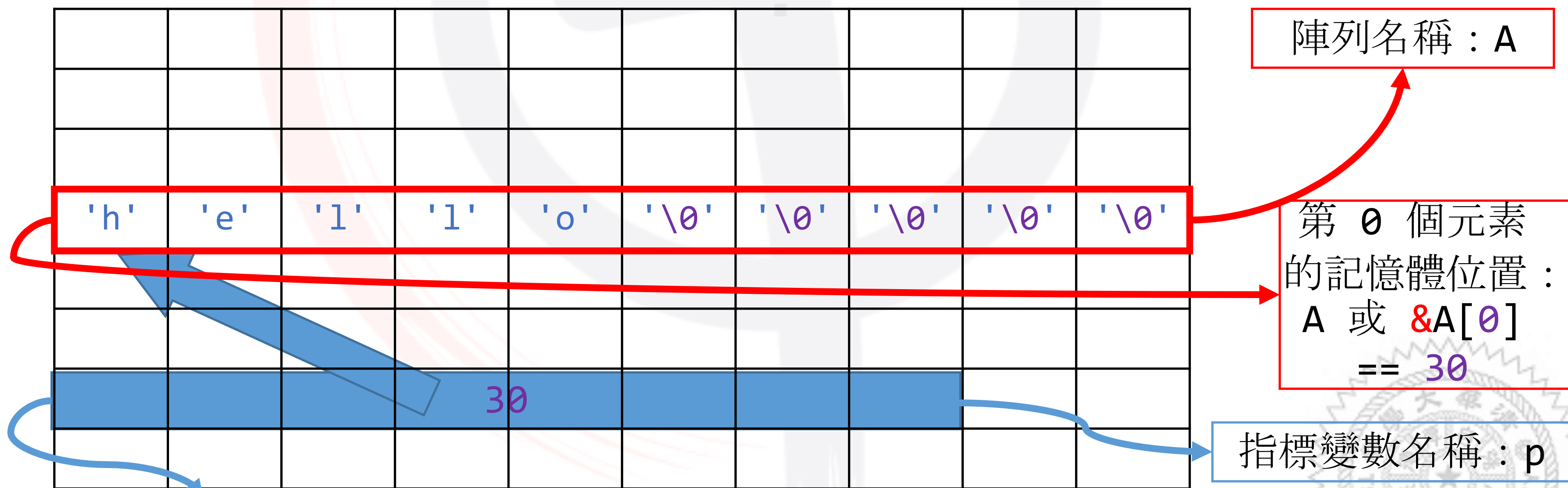
- 但如何連續宣告指標變數？
- `int* p, q, r; // 錯誤！`
- 這樣 p 是整數指標變數，但 q 跟 r 是整數變數！
- 要改成 `int *p, *q, *r;`

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

Relationship between Arrays and Pointers (1/2)

- 陣列名稱實際上就是指標！(也就是第 0 個元素的記憶體位置！)

```
char A[10] = "hello";
char *p = "hello";
p = A; /* p is assigned address of A[0] (就是 &A[0]) */
```



Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

Relationship between Arrays and Pointers (2/2)

- 陣列名稱實際上就是指標！(也就是第 0 個元素的記憶體位置！)

```
char A[10] = "hello";
char *p = "hello";
p = A; /* p is assigned address of A[0] (就是 &A[0]) */
```

- 差異？
  - sizeof(A) == 10 chars == 10 bytes
  - 但不管 p 是什麼指標(int*, char*…)且不管 p 指到什麼
    - sizeof(p) == 4 bytes (32位元機器) 或 8 bytes (64位元機器)
  - 可以改變指標變數存的值(記憶體位置) (L-value)
    - e.g., p = p + 1;
  - 不可以改變陣列儲存的記憶體位置(OS決定) (R-value)
    - e.g., A = A + 1;
  - 指標只記陣列的開頭位置，因此根本不知道陣列多大！

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

Pointer Variable vs. Array Notation (1/3)

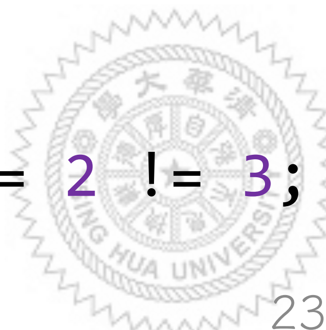
- 假設 A 是一個陣列， p 是一個指標變數

```
int A[10] = {1, 3, 5, 7, 9};
int *p;
p = A; /* p is assigned address of A[0] (就是 &A[0]) */
```

Expression	L-value	R-value
p	p	&p[0]
(p + n)	-	&p[n]
*p	p[0]	p[0]
*(p + n)	p[n]	p[n]

Expression	L-value	R-value
A	-	&A[0]
A + n	-	&A[n]
*A	A[0]	A[0]
*(A + n)	A[n]	A[n]

- 所以：
- `*p == p[0] == *A == A[0] == 1;`
- `*(p + 1) == p[1] == *(A + 1) == A[1] == 3;`
- `*p + 1 == p[0] + 1 == *A + 1 == A[0] + 1 == 2 != 3;`



Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

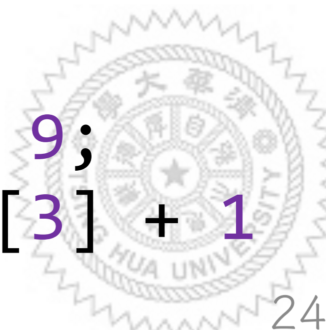
Pointer Variable vs. Array Notation (2/3)

- 假設 A 是一個陣列， p 是一個指標變數

```
int A[10] = {1, 3, 5, 7, 9};
int *p;
p = &A[3]; /* p is assigned address of A[3] */
```

Expression	L-value	R-value
(p - 3)	-	&p[-3] == &A[0]
(p - A)	-	3 // 差幾個元素的距離
(p + A)	非法	非法
(p * A)	非法	非法
(p / A)	非法	非法

- 所以，
- *p == p[0] == *(A + 3) == A[3] == 7;
- *(p + 1) == p[1] == *(A + 4) == A[4] == 9;
- *p + 1 == p[0] + 1 == *(A + 3) + 1 == A[3] + 1 == 8 != 9;



Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

Pointer Arithmetic (指標算術)

type	address	+	int offset	==	address
meaning	A	+	i	==	&A[i]
example	8	+	6	==	56

- 假設
  - double A[10]; double *p = A + 6;   sizeof(double) == 8;
- 則
  - p == A + 6 == 8 + 6*8 == 56 == &A[6]
  - p - A == (56 - 8) / 8 == 6

指標變數名稱：p

56

記憶體位置：&p == 88

陣列名稱：A

0	1	2	3	4	5	6	7	8	9
1.0	3.0	5.0	7.0	9.0	2.0	4.0	6.0	8.0	10.0

8	16	24	32	40	48	56	64	72	80

第 0 個元素的記憶體位置：A 或 &A[0] == 8

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

Printing pointers (addresses) (1/2)

- `int A[10] = {1, 3, 5, 7, 9}, *p = A + 6;`
- `cout << "p - A = " << p << " - " << A << " = " << p - A << endl;`

```
p - A = 0x22fe28 - 0x22fe10 = 6
```

- 0x22fe28 – 0x22fe10 == $18_{16} == 1\times16^1 + 8\times16^0 == 24$ (bytes)
- 24/sizeof(int) == 24/4 == 6

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

Printing pointers (addresses) (2/2)

- `char A[10] = "hello", *p = A + 3;`
- `cout << "p - A = " << p << " - " << A << " = " << p - A << endl;`

```
p - A = lo - hello = 3
```

- What's wrong?
  - 因為 char* 跟 char [] 的行為很像 (p 存位置， A 是 A[0] 之位置)
  - 所以 cout 把他們都當字串來印 (自動 cout << *p << *(p + 1) << ... 直到遇到結尾字元'\0'為止)
- 如何解決？
  - 把指標轉成 void* (type-casting 成指到 unknown type 的指標)
  - `cout << "p - A = " << (void*) p`
    `<< " - " << (void*) A << " = " << p - A << endl;`

```
p - A = 0x22fe23 - 0x22fe20 = 3
```

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

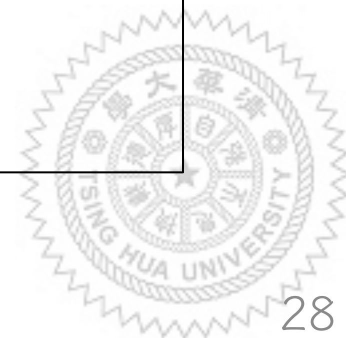
Revisited: Copying Arrays with known length

- Ch. 5 – Array version

```
void assignIntArray(int source[], int dest[], int size){
    for (int i = 0; i < size; i++)
        dest[i] = source[i];
}
```

- Here – Pointer version

```
// Rewrite in pointer syntax: No need for index variable i!
void assignIntArray(int *src, int *dest, int size){
    int *end = src + size;
    while (src < end)
        *dest++ = *src++;
}
```



Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

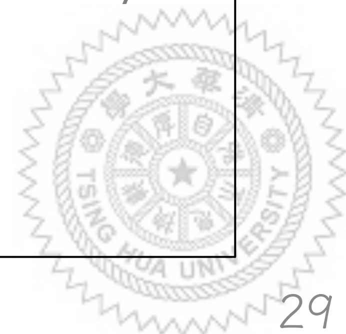
Revisited: Copying Null-terminated String

- Ch. 5 p.19 — Array version

```
void assignString(char source[], char dest[]){
    int i = 0;
    do {
        dest[i] = source[i];
    } while (source[i++]);
}
```

- Here — Pointer version

```
/* Rewrite in pointer syntax: copy content and post-
   increment both pointers until last copied char is null.*/
void assignString (int *src, int *dest){
    do {
    } while (*dest++ = *src++);
}
```



Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

Assignment to a pointer variable

- 指標變數：存記憶體位置
- 但資料在哪？

1. NULL: 沒有有效資料
2. Allocated statically (靜態分配)
   - 全域變數、static-local 變數：生存時間 = 整個程式啟動至結束
   - 常數 (e.g., 字串常數)：生存時間 = 整個程式啟動至結束
3. Allocated dynamically (動態分配)
   - Auto-local 變數：生存時間 = 只在該 block 期間生存 (stack frame)
   - new：生存時間 = 從 new 到 delete (heap)

取自高捷少女

危險 – 常見 bugs

x 試著修改 immutable 資料 (e.g., 字串常數)

x 試著回傳 local 變數！(local 變數消失後還使用它)

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

Examples of assignment to pointer variable

```
char myGlobal[23]; /* a global variable */
int main() {
    char *p = myGlobal; /* p gets static address -- ok */
    static char s; /* like a global but visible within main() */
    p = &s; /* also ok, since s has static address */
    p = "hello"; /* string literal also has static address */
    if (p) {
        char w[23] = "world";
        char *x = "world";
        p = x; /* this is ok because p copies the address of
                  a string literal (which is static) */

        p = w; /* Be careful! w no longer exists after { }
                  w's lifetime is shorter than p's lifetime! */
    }
    cout << "w is dead, "
         << "w's location will be occupied by others.\n";
}
```

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

Revisited: String Library

- C 字串: 使用 strcpy() or strncpy()
  - #include <cstring>
  - int strlen(char *s); – 回傳字串 s 的長度
  - char *strcpy(char *s1, char *s2);
    - 複製字串 s2 給 s1，直到遇到 s2 的 '\0' 字元
    - 回傳值就是 s1 的地址 (R-value)
  - char *strncpy(char *s1, char *s2, size_t n);
    - 複製字串 s2 給 s1，直到遇到 s2 的 '\0' 字元或直到複製 n 個字為止
    - 常使用在 s1 空間比 s2 小的時候
- C++ 字串：直接用 = …
  - #include <string>
  - s.length(); – 回傳字串 s 的長度
  - s1 = s2; – 複製字串 s2 給 s1，直到遇到 s2 的 '\0' 字元
- 其他陣列型態：使用 memcpy()
  - #include <cstring>
  - void memcpy(void *s1, void *s2, size_t nBytes);

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

OK to return (the address of) string literals

```
char *RPS(int n){
    switch (n){
        case 0:
            return "scissors";
        case 1:
            return "rock";
        case 2:
            return "paper";
    }
}
int main(){
    srand(time(NULL));
    char *p;
    int rounds = 10;
    while (rounds--)
        cout << (p = RPS(rand() % 3)) << endl;
}
```

這樣回傳的是字串常數的位置(字串第1個字的記憶體位置)，而非內容。

```
paper
rock
scissors
paper
scissors
scissors
scissors
rock
rock
scissors
```

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

Do NOT return a pointer to an auto-local variable

```
char *buf = new char [20];
```

```
char *getInput(char *prompt) {
    char buf[20];
    cout << prompt;
    cin >> buf;
    return buf;
}
int main() {
    char *r = getInput("Please enter your name: ");
    cout << "your name is " << r << endl;
}
```

buf 的生存時間到這，一離開這， buf 的內容可能就會被別人所用！

r 拿到 buf 的位址，但 buf 的內容可能隨時被改。

- 解法：

1. (static) char buf[20]; (放在 global)
2. static char buf[20]; (local-static 生存時間是整個程式)
3. 動態給 buf 記憶體空間 (生存時間從 new 到 delete (heap))

```
char *buf = new char [20]; …; delete [] buf;
```


Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

Pointer Types

- 不能將某型別的變數的地址指派給另一個型別的指標變數！
  - 編譯器不讓你這麼做
  - 除非你明顯地做 type–casting

```
int i = 3;
char *q;
q = &i;       // illegal
q = (char*) &i;  // this forces p to be a char address!
```

- 例外：void pointer

```
void *p;
p = &i;
```

  - 這不代表 p 指向 void type 的變數，而是代表它可以是任意 type 的指標
  - 指標算術時， 1 個 void 是 1 個 byte

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

Revisited: pointer version of Sorting (1/4)

- Ch.5 p.38 — Array version

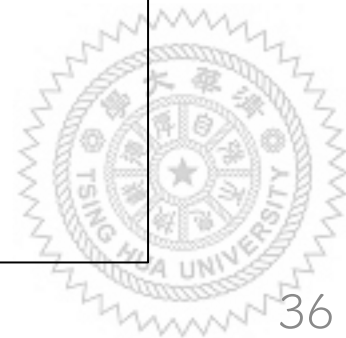
```
void selectionSort(int A[], int size) {
    for (int i = 0; i < size; i++) {
        int m = findMinIndex(A, i, size);
        swap(A, i, m);
    }
}
```

- Here — Pointer version

Actually is int *A

```
void selectionSort(int A[], int size) {
    while (size > 0){
        int *minPtr = findMinIndex(A, size--);
        swap(A++, minPtr);
    }
}
```

Why can we do A = A + 1;?



Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

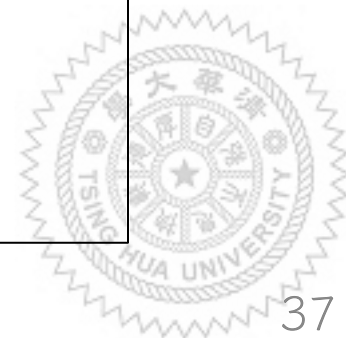
Revisited: pointer version of Sorting (2/4)

- Ch.5 p.38 — Array version

```
int findMinIndex(int A[], int start, int size) {
    int minIndex = -1;
    for (int i = start; i < size; i++)
        if (minIndex < 0 || A[i] < A[minIndex])
            minIndex = i;
    return minIndex;
}
```

- Here — Pointer version

```
int *findMinIndex(int A[], int size) {
    int *minPtr = NULL, *p;
    for (p = A, A += size; p < A; p++)
        if (minPtr == NULL || *p < *minPtr)
            minPtr = p;
    return minPtr;
}
```



Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

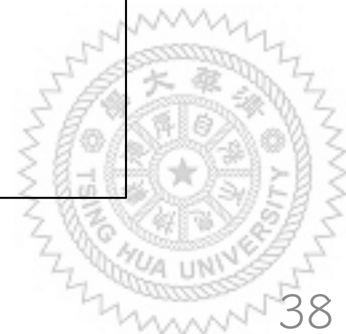
Revisited: pointer version of Sorting (3/4)

- Ch.5 p.38 – Array version

```
void swap(int A[], int x, int y){
    int temp = A[x];
    A[x] = A[y];
    A[y] = temp;
}
```

- Here – Pointer version

```
void swap(int *x, int *y){
    int temp = *x;
    *x = *y;
    *y = temp;
}
```



Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

Revisited: pointer version of Sorting (4/4)

- Ch.5 p.39

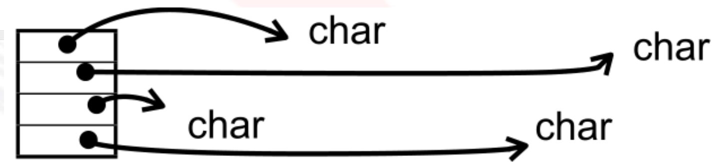
```
int main() {
    int A[100], size = 0, i = 0;
    while (size < 100 && cin >> A[size++]);
    selectionSort(A, --size); // Why do we do --size?
    while (i < size)
        cout << A[i++] << ' ';
}
```

```
1 10 11 2 3 7 5 9 4 4
^Z
1 2 3 4 4 5 7 9 10 11
```

Ctrl-Z + Enter to make this become `false`

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

Array of Pointers vs. Pointer to Array (指標陣列 vs. 陣列指標)



- `char *suit[4];`
- 這樣宣告了一個以 4 個 `char*`（字元指標）為元素的陣列
- 而不是宣告了一個指標指向一個以 4 個 `char` 為元素的陣列

- How about just `char *suit;`?
- 所以到底該怎麼辦？

```
typedef char FourCharArray[4];
FourCharArray *ptr; /* pointer to a 4-char array */
```

rmission required for reproduction or display.

Array of Pointers vs. Two-dimensional Array (指標陣列 vs. 二維陣列) (1/2)

```
char *suit[4] = { /* array of 4 pointers to strings */
    "Heart", "Diamond", "Club", "Spade"
};
```

- suit[0], …, suit[3] 都有 L-value 與 R-value (因為是char*)

所以可以 suit[0] = "hello";

- suit[1][0] == 'D'; suit[1][0] = 'C';
- 如果 suit[i] 指向 NULL, suit[i][j] 不存在。

```
char matrix[4][9] = { /* matrix of 4 rows of 9-char buffers */
    "Heart", "Diamond", "Club", "Spade"
};
```

- matrix[0], …, matrix[3] 只有 R-value (因為是指 &matrix[0][0], …, &matrix[3][0])
- matrix[1][0] == 'D'; ~~matrix[1][0] = 'C';~~

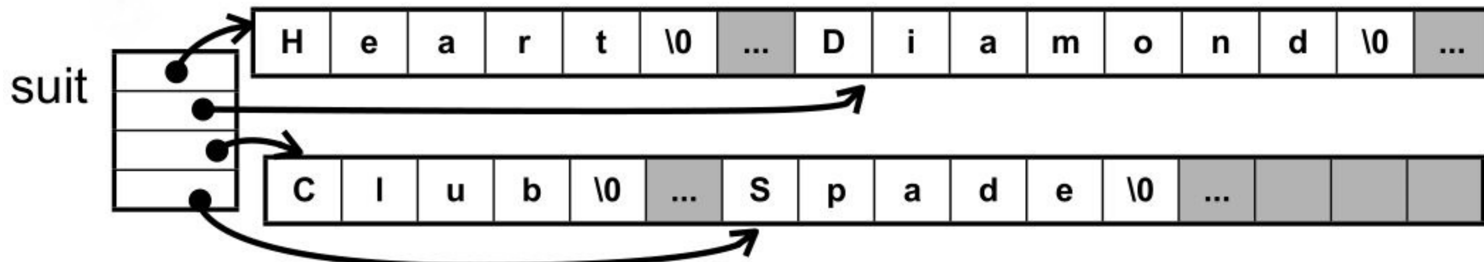
所以不可以 matrix[0] = "hello";

- matrix[i][j] 有 L-value (因為是一個 char)

Copyright © 2016 Slighten and CSTC. Permission required for reproduction or display.

Array of Pointers vs. Two-dimensional Array (指標陣列 vs. 二維陣列) (2/2)

```
char *suit[4] = { /* array of 4 pointers to strings */
    "Heart", "Diamond", "Club", "Spade"
};
```



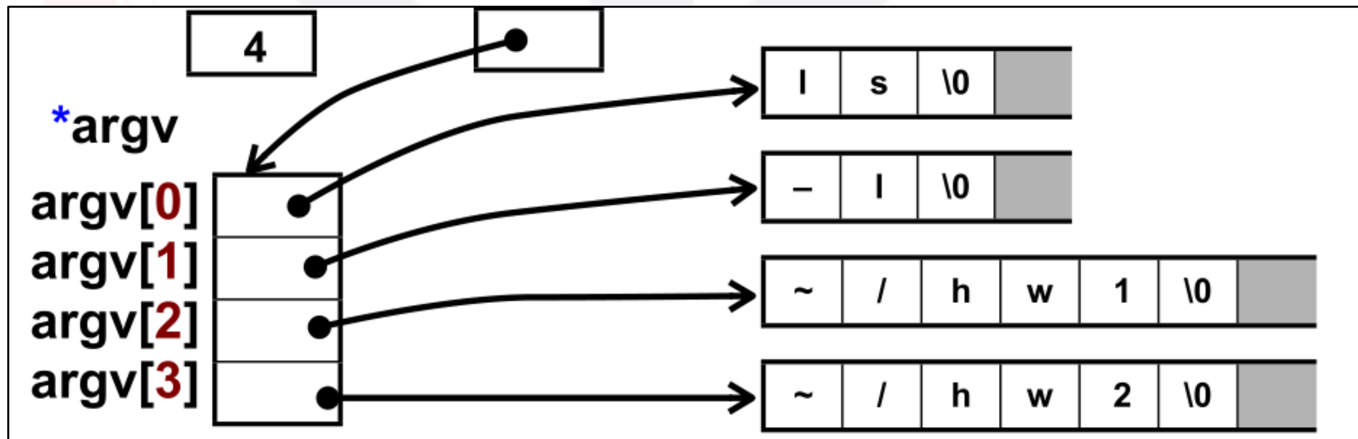

```
char matrix[4][9] = { /* matrix of 4 rows of 9-char buffers */
    "Heart", "Diamond", "Club", "Spade"
};
```

	0	1	2	3	4	5	6	7	8
0	H	e	a	r	t	\0	-	-	-
1	D	i	a	m	o	n	d	\0	-
2	C	l	u	b	\0	-	-	-	-
3	S	p	a	d	e	\0	-	-	-

quired for reproduction or display.

main(int argc, char **argv) revisited

- 回憶 main function 的參數
  - `int argc; /* argument count */`
  - `char **argv; /* argument list (pointer to strings) */`
  - 也可以宣告成 `char *argv[]; /* array of (char*)s */`
- 例：`% ls -l ~/hw1 ~/hw2`
- `int main(int argc, char **argv)`



Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

剩下 OOP 以前內容的介紹

遞迴 (Recursion) 簡介

- function call 自己就是遞迴
  - 當巨觀行為與微觀行為類似時，e.g., 碎形、河內塔…
  - 當每一步行為類似時 (費波那契、階乘、連加…)
- 有一個問題，何時結束?

```
int factorial(int n) {
    return (n <= 1) ? 1 : n * factorial(n-1);
}
```

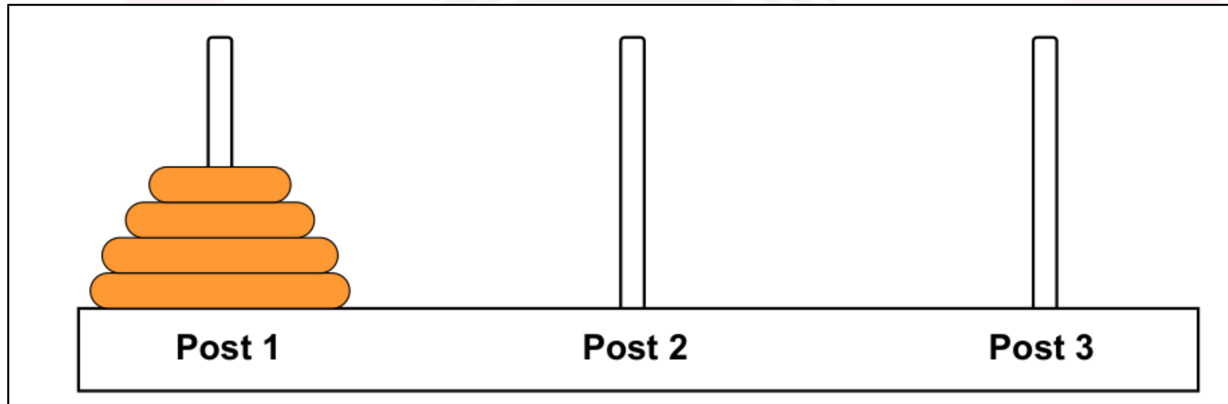
⇒需要有 base condition (終止條件)

```
// Recursive version
int factorial(int n) {
    if (n <= 1) return 1; // base condition
    return n * factorial(n-1);
}
```

```
// Iterative version
int factorial(int n) {
    int res = 1;
    while (n > 1) res *= n--;
    return res;
}
```

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

河內塔 (Tower of Hanoi)



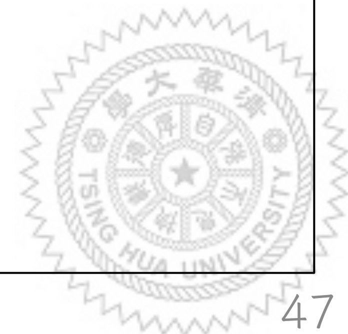
- 把 4 個盤子從 Post 1 搬到 Post 3：
  1. 先把 3 個盤子從 Post 1 搬到 Post 2
     1) 先把 2 個盤子從 Post 1 搬到 Post 3
        - …
     2) 把第 3 個盤子從 Post 1 搬到 Post 2
     3) 再把 2 個盤子從 Post 3 搬到 Post 2
  2. 把第 4 個盤子從 Post 1 搬到 Post 3
  3. 再把 3 個盤子從 Post 2 搬到 Post 3

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

河內塔程式碼

```
Move disk number 1 from 1 to 2.
Move disk number 2 from 1 to 3.
Move disk number 1 from 2 to 3.
Move disk number 3 from 1 to 2.
Move disk number 1 from 3 to 1.
Move disk number 2 from 3 to 2.
Move disk number 1 from 1 to 2.
Move disk number 4 from 1 to 3.
Move disk number 1 from 2 to 3.
Move disk number 2 from 2 to 1.
Move disk number 1 from 3 to 1.
Move disk number 3 from 2 to 3.
Move disk number 1 from 1 to 2.
Move disk number 2 from 1 to 3.
Move disk number 1 from 2 to 3.
```

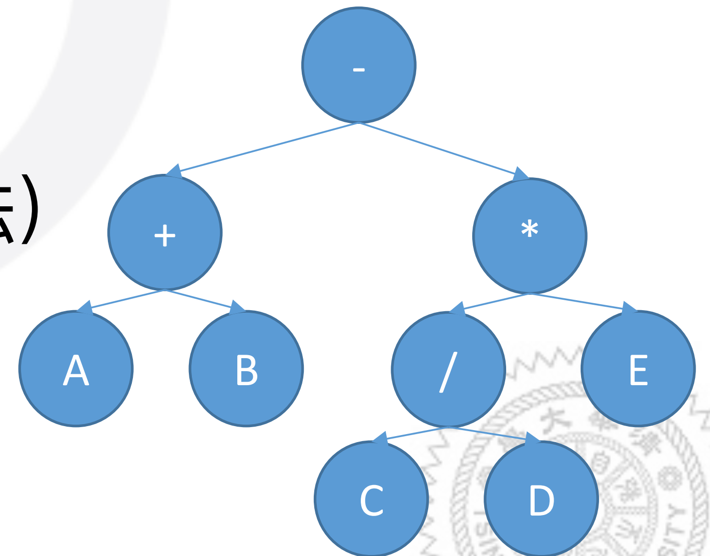
```
void moveDisk(int diskNumber, int startPost, int endPost, int midPost){
    if (diskNumber > 1) {
        /* Move top n-1 disks from start post to mid post */
        moveDisk(diskNumber-1, startPost, midPost, endPost);
        cout << "Move disk number " << diskNumber
             << " from " << startPost << " to " << endPost << endl,
        /* Move n-1 disks from mid post to end post */
        moveDisk(diskNumber-1, midPost, endPost, startPost);
    }
    else cout << "Move disk number 1 from " << startPost
              << " to " << endPost << endl;
}

int main() {
    moveDisk(4, 1, 3, 2);
}
```



Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

遞迴後續

- Binary Tree Traversal (二元樹的走訪)
  - (中左右)前序 (Pre–order): – + A B * / C D E
  - (左中右)中序 (In–order): (A + B) – ((C / D) * E)
  - (左右中)後序 (Post–order): A B + C D / E * –
- Prefix Calculator (+ 2 3 = 5)
- Divide and Conquer (演算法)
- Dynamic Programming (演算法)



Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

C 檔案讀寫(C File I/O) (1/3)

- 觀念：Each file is associated with a stream (串流).
- 步驟：建立檔案指標→開檔→讀寫資料→關檔
- C: type of stream = file handle (file pointer) (`FILE *`)
  - `#include <cstdio>`
  - 建立檔案指標：`FILE *fh;`
  - 開檔：`FILE *fopen(char *name, char *mode);`
    - name：檔名，mode：使用檔案的方式("r" = read, "w" = write, "a" = append)
    - 回傳 NULL，代表可能：
  1. 此檔案不存在
  2. 你無權限存取此檔案
      - 最好用 `if (!fh)` 判斷檔案是否開啟成功了
  - 讀檔：`fscanf(file handle, scanf() 參數形式);`
  - 寫檔：`fprintf(file handle, printf() 參數形式);`
  - (註：讀寫二進位資料：`fread()` and `fwrite()`)
  - 關檔：`int fclose(FILE*);`

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

C 檔案讀寫(C File I/O) (2/3)

```c
FILE *fin;                                        // 建立檔案指標
fin = fopen("filename.txt", "r");                 // 開檔
if (fin){                                         // if fin != NULL
    fscanf(fin, "%d%c%s", &num, &ch, str);        // 讀檔
    fclose(fin);                                  // 關檔
} else printf("Unable to open file\n");

/* Do some data manipulation */

FILE *fout;                                       // 建立檔案指標
fout = fopen("filename.txt", "w");                // 開檔
if (fout){                                        // if fout != NULL
    fprintf(fout, "num = %d, ch = %d, str = %s\n",
                           num, ch, str);         // 寫檔
    fclose(fout);                                 // 關檔
} else printf("Unable to open file\n");
```

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

C 檔案讀寫(C File I/O) (3/3)

- 進階概念：

緩衝式 I/O

  - 隨機檔案存取 (寫讀/寫哪就讀/寫哪) – fseek(), ftell()...
  - 出清串流 – fflush()
  - http://www.cplusplus.com/reference/cstdio/
- 檔案 I/O 與作業系統
  - 阻擋式 (blocking) 與非阻擋式 (non-blocking) I/O
  - 同步 I/O – fsync()
  - 直接 I/O
  - 多工式 I/O – select(), poll()
  - O'REILLY 《LINUX 系統程式設計》

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

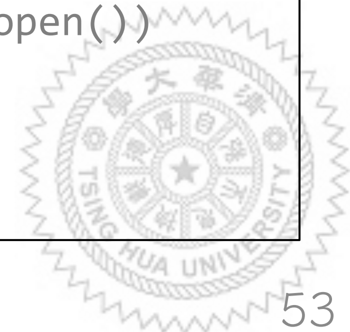
C++檔案讀寫 (C++ File I/O) (1/3)

- 觀念：Each file is associated with a stream (串流).
- 步驟：建立檔案指標→開檔→讀寫資料→關檔
- C++: type of stream = fstream, ifstream, ofstream
  - #include <fstream>
  - 建立檔案指標：ifstream fin; ofstream fout;
  - 開檔：void fstream::open(const char *name, ios::mode);
    - name：檔名，mode：使用檔案的方式(ios::in = read, ios::out = write, ios:app = append)
    - 回傳 NULL，代表可能：
      1. 此檔案不存在
      2. 你無權限存取此檔案
         - 最好用 if (!fin) 判斷檔案是否開啟成功了
  - 讀檔：
    - fin >> buf;
    - 一次一行：istream& getline(char* data, int length);
istream& getline(istream& is, string& str);
    - 一次一字元：istream& get(char& c);
  - 寫檔：
    - fout << buf;
    - 一次一字元：istream& put(char c);
  - (註：讀寫二進位資料：read() and write())
  - 關檔：void close();

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

C++檔案讀寫 (C++ File I/O) (2/3)

```
ifstream fin;                                    // 建立檔案指標
fin.open("filename.txt", ios::in);               // 開檔
// 可合併成 ifstream fin("filename.txt",ios::in);
if (fin){                                        // if (fin.is_open())
    string buffer;
    while(getline(fin, buffer))                  // 讀檔
        cout << buf << endl;
    fin.close();                                 // 關檔
} else cout << "Unable to open file" << endl;

/* Do some data manipulation */

ofstream fout;                                   // 建立檔案指標
fout.open("filename.txt", ios::out);             // 開檔
// 可合併成 ofstream fout("filename.txt",ios::out);
if (fout) {                                      // if (fout.is_open())
    fout << "This is a line.\n";                 // 寫檔
    fout.close();                                // 關檔
}
```



Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

C++檔案讀寫 (C++ File I/O) (3/3)

- 進階概念：

緩衝式 I/O

  - 隨機檔案存取 (寫讀/寫哪就讀/寫哪) – seekg(), tellg(), seekp(), tellp()…
  - 出清串流 – flush()
  - http://www.cplusplus.com/reference/fstream/fstream/
- 檔案 I/O 與作業系統
  - 阻擋式 (blocking) 與非阻擋式 (non-blocking) I/O
  - 同步 I/O – fsync()
  - 直接 I/O
  - 多工式 I/O – select(), poll()
  - O'REILLY 《LINUX 系統程式設計》

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

結構 (struct) 簡介 (1/5)

將不同 type 的相關 data 包在一起！

- Definition

```
struct struct_name {
    char a[10];
    int b;
    double c;
};
```

- Declaration

```
struct struct_name variable_name;
```

- Definition & Declaration at Once

```
struct struct_name {
    char a[10];
    int b;
    double c;
} variable_name;
```

- Struct Reference (用運算子 '.')

```
variable_name.a[0] = 'c';
variable_name.b = 10;
variable_name.c = 3.5;
```

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

結構 (struct) 簡介 (2/5)

- 可以用 `typedef` 使 code 好讀
- Syntax:

```
typedef type name;
```

- 例：

```
typedef int Color;
typedef struct flightType Flight;
typedef struct ab_type {
    int a;
    double b;
} ABGroup;
```

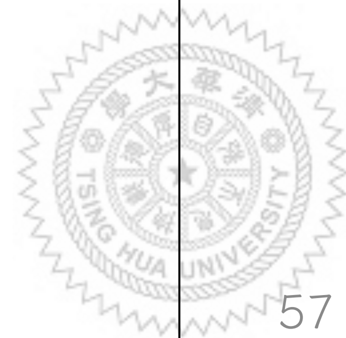
- 使用：

```
Color pixel[500]; // int pixel[500];
Flight plane1, plane2; // struct flightType plane1, plane2;
```

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

結構 (struct) 簡介 (3/5)

```
typedef struct POS{
    int x, y;
} Pos;

struct CHARACTER{
    string name;
    string weapon;
    Pos pos;
    unsigned int level, exp;
    int health_power;
    int attack_dmg;
    int defence_power;
} mainCharacter;

int main() {
    struct CHARACTER csw = {
        "CSW", "belly", {1, 3}, 1, 0, 100, 50, 10
    };
    mainCharacter = {
        "Slighten", "mouth", {2, 4}, 99, 0, 9999, 100, 100
    };
```



Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

結構 (struct) 簡介 (4/5)

```cpp
    mainCharacter.weapon = "handsome";
    mainCharacter.pos.y--;
    cout << mainCharacter.name << " vs. "
         << csw.name << endl;
    cout << "level: " << mainCharacter.level
         << " vs. " << csw.level << endl;
    cout << "weapon: " << mainCharacter.weapon
         << " vs. " << csw.weapon << endl;
    struct CHARACTER *myPtr = &mainCharacter;
    (*myPtr).weapon = "knife";
    myPtr->attack_dmg = 150;
}
```

```
Slighten vs. CSW
level: 99 vs. 1
weapon: handsome vs. belly
```

- myPtr 是一個結構指標
- (*myPtr).name == myPtr->name != *myPtr.name

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

結構 (struct) 簡介 (5/5)

- myPtr 是一個結構指標
- (*myPtr).name == myPtr->name != *myPtr.name == *(myPtr.name)
- 因為 . 的運算比 * 先！
- 另外

1. struct 有 = (assign) 運算子 (copy value)

⇒e.g., mainCharacter = csw;

2. struct* 還有 ==, != 運算子
3. 也可以宣告結構陣列：struct CHARACTER character[100];
4. union – 同樣資料可以用不同 type 存取
5. 動態記憶體配置
6. linked list –– built from struct and dynamic allocation

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

動態配置記憶體簡介 (1/2)

- 因為 local 變數一離開 {} 就會消失 (local 變數的記憶體位置一離開 {} 就會有可能被其他人使用 (存在 stack 裡))，因此 function 不應回傳 local 變數！
- 那還是想回傳 local 變數怎麼辦？

⇒動態給它記憶體 (存在 heap 裡)，回傳後該記憶體位置仍存在

⇒生存時間從 `new` 到 `delete`

- Syntax:

```
type* var_name = new type[個數];
delete [] var_name;
```

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

動態配置記憶體簡介 (2/2)

```
char *foo(){
    char *p = new char [10];
    strcpy(p, "hello\n");
    return p;
}

int main() {
    char *ptr = foo();
    cout << ptr;
}
```

Wrong!

```
char *foo(){
    char p[10];
    strcpy(p, "hello\n");
    return p;
}

int main() {
    char *ptr = foo();
    cout << ptr;
}
```

- 補充：當想要宣告長度超長的陣列時，也需要動態配置！
- 例：`int p[1000000]; // GG! Not so much space in the stack.`
- 改成：`int *p = new int [1000000];`

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

補充：字串在記憶體中的配置

```
char *globalHeader = "Chapter";
char globalArrayHeader[] = "Chapter";

void displayHeader(){
    static char* staticHeader = "Chapter";
    static char staticArrayHeader[] = "Chapter";
    char* localHeader = "Chapter";
    char localArrayHeader[] = "Chapter";
    char* heapHeader =
        (char*) malloc(strlen("Chapter")+1);
    strcpy(heapHeader, "Chapter");
}
```

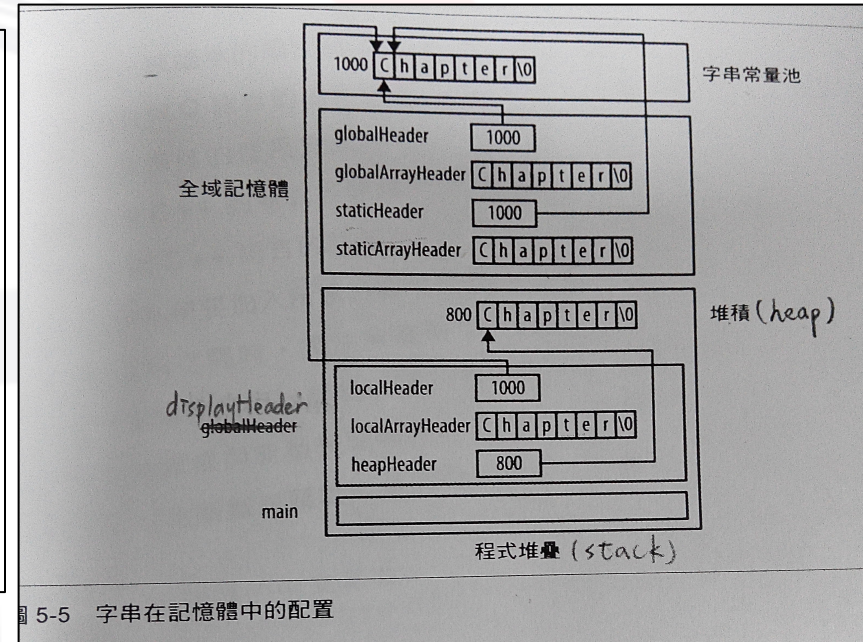


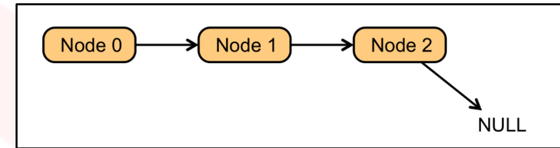
圖 5-5 字串在記憶體中的配置

取自歐萊禮出版的書

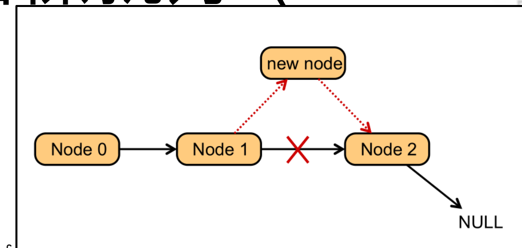
```
xxxHeader[2] = 'b'; // illegal
xxxArrayHeader[2] = 'b'; // legal
heapHeader[2] = 'b'; // legal
```

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

Linked List 簡介



- linked list 是把一個一個 node 用 pointer 連起來的資料結構
- 想像如果有一個 `int array[100];`
- 如果有資料要「插入」第 0 個，那是不是要把原本第 0~99 每個都往後搬一個 (就好像排隊有人插隊，所有人都要往後移動)
- 這樣需要移動資料的方式會相當的「耗時」！
- 想像排隊的不是人，而是上億萬篇網頁，如果是用陣列會完蛋！
- 相對的，linked list 只需要改指標就好 (recall 一個指標只有 4 或 8 bytes)！



Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

Linked List 簡介

(Let n = size)	ArrayList	LinkedList
add(e)	O(1)W	O(1)W
add(index, e)	O(n)W	**O(n)R + O(1)W**
get(index)	**O(1)R**	O(n)R
contains(e)	O(n)R	O(n)R
remove(e)	O(n)R + O(n)W	O(n)R + O(1)W
remove(index)	O(n)W	**O(n)R + O(1)W**

- Array is good for read – intensive access
  1. 總數固定時
  2. 時常查找與更改元素內容時
- Linked List is good for frequent structure modification
  1. 常增加元素時
  2. 大小 (總數) 常改變時

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

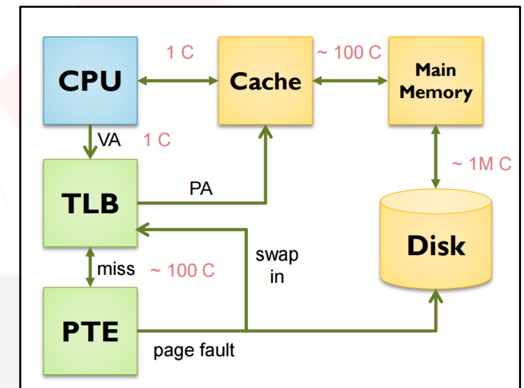
剩下的一些東西

1. 偶爾會用但沒提到的型別：enum（列舉）、union（聯集）
2. 開始寫大一點的 project (open source)
   - 100 行 → 500 行 → 1000 行 → 2000 行 …
   - 貪食蛇、俄羅斯方塊、自製遊戲…
   - 反組譯器→組譯器→CPU模擬器→編譯器…
3. 往後應該學的東西：
   1) 物件導向程式設計 (OOP, Object-Oriented Programming) (軟體實驗)
      - class, object, overload, inheritance, override, polymorphism, generic…
   2) C++ 標準模板庫 (STL, **Standard Template Library**)
      - <vector><list><queue><stack> <set><map><hash_map><algorithm><utility>...
   3) 資料結構→演算法→高等程式設計→…(平行演算法、隨機演算法、近似演算法…)
   4) 數位邏輯設計→計算機結構→作業系統→嵌入式系統→…(高等計組、平行程式…)
   5) 3) + 4) 電腦輔助設計概論→…(電子設計自動化、超大型積體電路量產可行性設計…)

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

組合語言簡介 (Assembly Language) (1/3)

- 組合語言：像機器碼的可讀式版本
- 指令集 (ISA, Instruction Set Architecture)：CPU 可做的行為
- 記憶體 (memory) vs. 快取 (Cache) vs. 暫存器 (registers, in CPU)



Memory (RAM)

Disk (ROM)

Cache

ALU

CPU

Input Devices

Output Devices

Registers

Control Unit

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

組合語言簡介 (2/3)

```
;
; Program to multiply a number by the constant 6
;
        .ORIG   x3050
        LD      R1, SIX
        LD      R2, NUMBER
        AND     R3, R3, #0      ; Clear R3.  It will
                                ; contain the product.
; The inner loop
;
AGAIN   ADD     R3, R3, R2
        ADD     R1, R1, #-1     ; R1 keeps track of
        BRp     AGAIN           ; the iteration.
;
        HALT
;
NUMBER  .BLKW   1
SIX     .FILL   x0006
;
        .END
```

- 以 LC-3 舉例
- 暫存器 8 個 (R0 ~ R7)
- 指令集的操作碼有：ADD, NOT, AND, JMP (jump), …
- 組語格式： (LABEL) OPCODE OPERANDS ; COMMENTS
- 一行一行執行
- OPCODE 就是操作碼，每個操作碼是一組獨立的數字
  - e.g., ADD = 0001, NOT = 1001, AND = 0101, …
- OPERAND 就是操作子 (R0~R8)
- 將機器碼 16 個一組(再幾個幾個一分割)
- 例如這行陳述式：ADD R3, R3, R2 ; R3 = R3 + R2
- 就會被翻譯成：0001 — 011 — 011 — 0 — 00 — 010

Opcode(4)	Dst(3)	Src1(3)	imm(1)	00	Src2(3)

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.

組合語言簡介 (3/3)

- Instruction 有 3 類
- 算術運算指令：ADD, AND, NOT
  - 沒有 SUB? MUL?：加負數就是減法、連加就是乘法
- 資料搬動指令：LD (load), ST (store)…
  - LD 把資料從記憶體搬到暫存器
  - ST 把資料從暫存器搬到記憶體
- 控制指令：JMP (jump), BR (branch), RTI (return), TRAP (system call)
  - JMP 有點像 function call
  - BR 有點像 if-else, while, … 的那些 test 判斷
- 編譯器將程式語言翻成組合語言
- 組譯器將組合語言翻成機器碼
- (反組譯器將機器碼翻回組合語言)

Copyright © 2016 Slighten and GOTC. Permission required for reproduction or display.