

程式設計

Ch09. Structure and Bitwise Operation

Chuan-Chi Lai 賴傳淇

Department of Communications Engineering
National Chung Cheng University

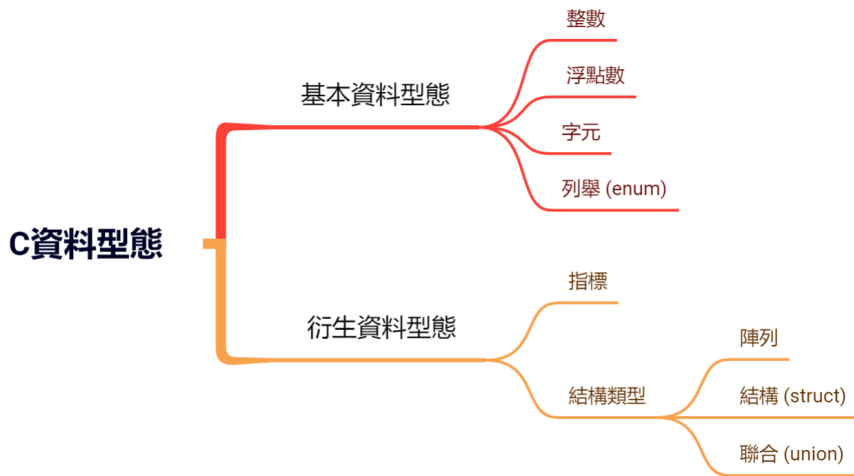
Spring Semester, 2024

Outline

- 1 衍生資料型別 (Derived Data Type)
- 2 typedef 宣告 (typedef Declarations)
- 3 結構的定義與宣告 (Structure Definition and Declaration)
- 4 結構的運算與成員的存取 (Structure Member Access)
- 5 結構的對齊與位元欄位 (Structure Member Alignment and Bit Fields)
- 6 Union 資料型別 (Union Data Type)
- 7 位元運算 (Bitwise Operation)
- 8 列舉資料型態 (Enumeration Data Type)
- 9 附錄：Base64

衍生資料型別 Derived Data Type

衍生資料型別 (Derived Data Type)



衍生資料型別 (Derived Data Type)

- 衍生資料型別是由其他型別為基礎來建構的，例如我們先前學過的指標與陣列。我們可以建立任何資料型別的指標與陣列。
- 而本章節要介紹的衍生資料型別——結構 (struct)、聯合 (union) 可由複數個不同的資料型別組成。
- 在章節的最後則會介紹類似於整數常數的列舉 (enum) 型別。

typedef 宣告 typedef Declarations

typedef 宣告 (typedef Declarations)

- typedef 宣告是將識別字宣告為新的資料型別的宣告式。使用 typedef 宣告可以將既有的資料型別取“別名”。

```
typedef DECLARATION;
```

- 將 typedef 後方加上宣告式，宣告識別字為該類別的別名。

typedef 宣告 (typedef Declarations)

- 建立 long long 的別名並宣告一個變數：

```
1 typedef long long LL; // 宣告 LL 為 long long 的別名
2 LL foo; // 宣告 LL(long long) 變數 foo
```

- 建立 char 相關的一系列別名：

```
1 typedef char Chr, *Str, ChrArray[100];
2 // 宣告 Chr 為 char 的別名
3 // 宣告 Str 為 char * 的別名
4 // 宣告 ChrArray 為 char[100] 的別名
```


typedef 宣告 (typedef Declarations)

- 使用 typedef 可以將名稱很長的變數型別進行簡化，讓程式寫作更有效率。
- 我們通常將 typedef 宣告的識別字第一個字母大寫，表示這個名稱是自行建立的型別名稱。

結構的定義與宣告 Structure Definition and Declaration

結構的定義與宣告 (Structure Definition and Declaration)

- 結構 (structure) 是個可以包含數個其他不同資料型別的資料型別，定義結構的範例如下：

```
1 struct student
2 {
3     char *name;
4     int grade;
5 };
```

- 在上述定義中，student 為結構標籤 (tag)，name 與 grade 為該結構的成員 (members)。

結構的定義與宣告 (Structure Definition and Declaration)

- 注意結構的成員不可以包含與自己相同的型別或陣列：

```
1 struct student
2 {
3     char *name;
4     int grade;
5     struct student bestFriend; // ERROR
6 };
```

結構的定義與宣告 (Structure Definition and Declaration)

- 結構的成員可以包含與自己相同的指標或指標陣列，這種結構稱為自我參考結構：

```
1 struct student
2 {
3     char *name;
4     int grade;
5     struct student *bestFriend; // OK
6 };
```

宣告結構變數

- 可以於結構定義時一並宣告，也可在之後宣告：

```
1 struct student
2 {
3     char *name;
4     int grade;
5 } stdntA, stdntLst[10];
```

```
1 struct student
2 {
3     char *name;
4     int grade;
5 };
6 struct student stdntA, stdntLst[10];
```

初始化結構變數

- 如同陣列宣告時的初始化，使用初始值串列 (大括號) 指派。

```
1 struct student
2 {
3     char *name;
4     int grade;
5 };
6 struct student foo = {"Trent", 1};
7 // 於 C99 以後也可指定成員來設定初始值
8 struct student bar = {.grade=2, .name="Jason"};
```

使用 typedef 宣告結構別名

- 可以於結構定義時一並宣告，也可在之後宣告：

```
1 typedef struct student
2 {
3     char *name;
4     int grade;
5 } Student;
6 Student foo; //宣告結構變數 foo
```

```
1 struct student
2 {
3     char *name;
4     int grade;
5 };
6 typedef struct student Student;
7 Student foo; //宣告結構變數 foo
```


結構的定義與宣告 (Structure Definition and Declaration)

- 於結構定義時一並宣告別名，其結構標籤 (tag) 可以被省略。

```
1 typedef struct
2 {
3     char *name;
4     int grade;
5 } Student;
6
7 Student foo; //宣告結構變數 foo
```

結構的運算與成員的存取 Structure Member Access

結構成員運算子 (.)

- 結構的成員需使用結構成員運算子 (.) 或稱點號運算子存取。
- 結構成員運算子會經由結構變數名稱來存取指定的結構成員。

```
1 typedef struct {  
2     char name[30];  
3     int grade;  
4 } Student;  
5  
6 int main() {  
7     Student foo; //宣告結構變數 foo  
8     scanf("%s%d", foo.name, &foo.grade);  
9     printf("%s grade:%d\n", foo.name, foo.grade);  
10 }
```

結構指標運算子 (->)

- 結構指標成員的存取需使用結構指標運算子 (->) 或稱箭號運算子。
- 運算式 `ptr->mem` 等同於 `(*ptr).mem`。

```
1 typedef struct {  
2     char name[30];  
3     int grade;  
4 } Student;  
5  
6 void inputStudent(Student *s) {  
7     scanf("%s%d", s->name, &s->grade);  
8 }
```

結構運算子 (. 與->) 常見的錯誤

- 結構指標運算子 (->) 的-與 > 之間不可有空白，此為語法錯誤。
- 結構指標運算子 (->) 的前後請勿加上空白，以免閱讀程式時誤認結構指標與成員為兩獨立變數名稱。
- 結構指標運算子 (->) 與結構成員運算子 (.) 的優先度高於 * 與 &，因此 (*ptr).mem 與 *ptr.mem 是不相同的。

結構的運算與成員的存取 (Structure Member Access)

結構變數可接受的運算：

- 同樣型別的結構變數互相指派 (=)。
- 取得結構變數的位址 (&)。
- 存取結構變數成員 (.)。
- 使用 sizeof 運算子計算結構變數的大小。

* 結構變數未取得成員時，不可進行算術、關係、邏輯等運算。

結構的對齊與位元欄位 (Structure Member Alignment and Bit Fields)

結構的對齊與位元欄位

Structure Member Alignment and Bit Fields

結構的對齊與位元欄位 (Structure Member Alignment and Bit Fields)

- 在計算結構大小时，常常會出現實際大小比結構成員的大小總和還多的情況，這是因為結構的對齊規則。

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef struct {
5      char a, b;
6      int i;
7  } Foo;
8
9  int main()
10 {
11     printf("size of Foo: %d\n", sizeof(Foo));
12 }
```

C:\Projects\ch09_code\struct_ x + v

size of Foo: 8

結構的對齊與位元欄位 (Structure Member Alignment and Bit Fields)

結構的對齊規則 (alignment) 包含以下幾點：

- ① 每個成員相對於第一個成員的偏移量 (offset) 必為該成員大小的倍數。
- ② 結構的大小 (size of) 必為結構成員中，型別最寬的成員大小的倍數。
- ③ 結構內部會填充空白位元 (padding) 以滿足上述要求。

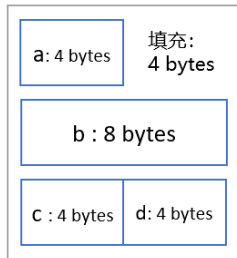
結構的對齊與位元欄位 (Structure Member Alignment and Bit Fields)

- 每個成員的偏移量 (offset) 必為該成員大小的倍數。

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef struct
5 {
6     int a;
7     long long b;
8     int c, d;
9 } Foo;
10
11 int main()
12 {
13     Foo foo;
14     printf("size of Foo: %d\n", sizeof(Foo));
15     printf("offset of 'a': %d\n", (void*)&foo.a - (void*)&foo);
16     printf("offset of 'b': %d\n", (void*)&foo.b - (void*)&foo);
17     printf("offset of 'c': %d\n", (void*)&foo.c - (void*)&foo);
18     printf("offset of 'd': %d\n", (void*)&foo.d - (void*)&foo);
19 }
```

C:\Projects\ch09_code\struct_ x +

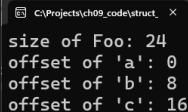
size of Foo: 24
offset of 'a': 0
offset of 'b': 8
offset of 'c': 16
offset of 'd': 20



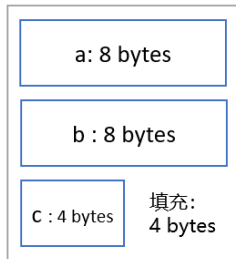
結構的對齊與位元欄位 (Structure Member Alignment and Bit Fields)

- 結構的大小 (size of) 必為結構成員中，型別最寬的成員的大小的倍數。

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef struct
5 {
6     long long a, b;
7     int c;
8 } Foo;
9
10 int main()
11 {
12     Foo foo;
13     printf("size of Foo: %d\n", sizeof(Foo));
14     printf("offset of 'a': %d\n", (void*)&foo.a - (void*)&foo);
15     printf("offset of 'b': %d\n", (void*)&foo.b - (void*)&foo);
16     printf("offset of 'c': %d\n", (void*)&foo.c - (void*)&foo);
17 }
```



```
size of Foo: 24
offset of 'a': 0
offset of 'b': 8
offset of 'c': 16
```



結構的對齊與位元欄位 (Structure Member Alignment and Bit Fields)

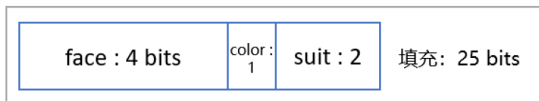
位元欄位 (bit fields)

- C 語言的結構中，我們可以指定 unsigned int 或 int 成員所佔的位元數量，將資料存放在最少的位元裡，以提高記憶體的使用率。

結構的對齊與位元欄位 (Structure Member Alignment and Bit Fields)

- 我們可以在成員名稱之後加上冒號 (:) 以及代表位元寬度的整數常數。此常數必須介於 0 至 int 所佔的位元個數 (32 bits) 之間。
- 例如我們可以使用以下結構來儲存一張撲克牌的資訊：

```
1 typedef struct
2 {
3     unsigned face : 4;
4     unsigned color : 1;
5     unsigned suit : 2;
6 } Card;
```



結構的對齊與位元欄位 (Structure Member Alignment and Bit Fields)

- 一個位元寬度為 N 的 unsigned int 結構成員，其可儲存的變數範圍為 0 到 $2^N - 1$ 。
- 一個位元寬度為 N 的 int 結構成員，其可儲存的變數範圍為 -2^{N-1} 到 $2^{N-1} - 1$ 。
- 例如上頁的 unsigned int 結構成員 face 佔用了 4 個位元，儲存範圍是 0 至 15，足夠用來儲存撲克牌的 Ace 至 King(共 13 種)。
- 結構成員 suit 佔用了 2 個位元，儲存範圍是 0 至 3，也足夠用來儲存撲克牌的 4 種花色。

結構的對齊與位元欄位 (Structure Member Alignment and Bit Fields)

- 使用不具名的欄位，可以填補位元 (padding)，調整成員的偏移量。

```
1 typedef struct
2 {
3     unsigned face : 4;
4     unsigned suit : 2;
5     unsigned : 2;
6     unsigned color : 1;
7 } Card;
```



結構的對齊與位元欄位 (Structure Member Alignment and Bit Fields)

- 使用寬度為 0 的不具名的欄位，可以填補位元 (padding) 至下一個儲存單元 (通常是以 32 bits 為單位)。

```
1 typedef struct
2 {
3     unsigned face : 4;
4     unsigned suit : 2;
5     unsigned : 0;
6     unsigned color : 1;
7 } Card;
```



Union 資料型別 Union Data Type

Union 資料型別 (Union Data Type)

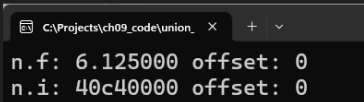
- union 與 struct 都是一種衍生的資料型別，但 union 的成員會共用相同的空間。
- 以下是 union 的定義範例。

```
1 union number
2 {
3     float f;
4     int i;
5 };
```

Union 資料型別 (Union Data Type)

- union 的成員會共用相同的記憶體空間，並依照該成員的型態解讀所佔記憶體空間中的內容。

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef union
5  {
6      float f;
7      int i;
8  } Number;
9
10 int main()
11 {
12     Number n;
13     n.f = 6.125;
14     printf("n.f: %f offset: %d\n", n.f, (void*)&n.f - (void*)&n);
15     printf("n.i: %x offset: %d\n", n.i, (void*)&n.i - (void*)&n);
16 }
```



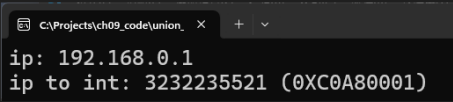
The debugger window shows the following memory addresses and offsets:

Variable	Address	Offset
n.f	6.125000	offset: 0
n.i	40c40000	offset: 0

Union 資料型別 (Union Data Type)

- 以下是環境為 Little-Endian 時，使用 union 將 ipv4 轉換成整數的範例：

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef union
5  {
6      unsigned char bytes[4];
7      unsigned int i;
8  } IPv4;
9
10 int main()
11 {
12     IPv4 ip = {.bytes = {[3] = 192, [2] = 168, [1] = 0, [0] = 1}};
13     printf("ip: %hhd.%hhd.%hhd.%hhd\n",
14         ip.bytes[3], ip.bytes[2], ip.bytes[1], ip.bytes[0]);
15     printf("ip to int: %u (%#X)\n", ip.i, ip.i);
16 }
```



位元運算 Bitwise Operation

位元運算 (Bitwise Operation)

- 所有資料在電腦裡都是以一連串的位元來表示，每個位元 (bit) 的值可為 0 或 1。
- 在大部分的系統裡，8 個連續的位元構成一個位元組 (byte)，也就是 C 語言中 char 型態所使用的儲存單位。
- 而其他的資料型別 (如 short、int、long long) 所儲存的資料則會存放在更多的位元組內。
- 位元運算元可以用來操作整數類型的運算元中的位元。

位元運算子 (1/2)

名稱	符號	說明
bitwise AND	&	若兩運算元同個位置的位元 (bit) 都為 1，則運算結果的同位置位元為 1，否則為 0。
bitwise OR		若兩運算元同個位置的位元至少有一為 1，則運算結果的同位置位元為 1，否則為 0。
bitwise XOR	^	若兩運算元同個位置的位元只有一為 1，則運算結果的同位置位元為 1，否則為 0。

位元運算子 (2/2)

名稱	符號	說明
left shift	\ll	例如 $x \ll n$: 將運算元 x 往左平移 n 個位元 (bit)，右邊以 0 填滿。
right shift	\gg	例如 $x \gg n$: 將運算元 x 往右平移 n 個位元 (bit)，左邊位元填空方是依不同環境而異。
complement	\sim	例如 x : 所有為 0 的位元數都設定為 1，所有為 1 的位元數都設定為 0。

位元運算 (Bitwise Operation)

- 印出整數的二進位：

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     unsigned input;
7     printf("Enter a positive number: ");
8     scanf("%i", &input);
9     printf("binary: ");
10    for (int i = 0; i < 32; i++)
11    {
12        unsigned offset = 31 - i;
13        printf("%d", (input & (1 << offset)) >> offset);
14        if (i && !(i+1) % 8) printf(" ");
15    }
16    puts("");
17 }
```

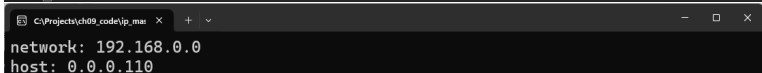
C:\Projects\ch09_code\unsign x + v

Enter a positive number: 987654321
binary: 00111010 11011110 01101000 10110001

位元運算 (Bitwise Operation)

- 使用位元運算操作 IPv4 遮罩：

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef union
5  {
6      unsigned char bytes[4];
7      unsigned int i;
8  } IPv4;
9
10 int main()
11 {
12     IPv4 myip = {.bytes = {[3] = 192, [2] = 168, [1] = 0, [0] = 110}};
13     IPv4 mask = {.bytes = {[3] = 255, [2] = 255, [1] = 255, [0] = 0}};
14     IPv4 network, host;
15     network.i = myip.i & mask.i;
16     host.i = myip.i & ~mask.i;
17     printf("network: %hhd.%hhd.%hhd.%hhd\n",
18         network.bytes[3], network.bytes[2], network.bytes[1], network.bytes[0]);
19     printf("host: %hhd.%hhd.%hhd.%hhd\n",
20         host.bytes[3], host.bytes[2], host.bytes[1], host.bytes[0]);
21 }
```



The screenshot shows a terminal window with the following output:

```
network: 192.168.0.0
host: 0.0.0.110
```

列舉資料型態 Enumeration Data Type

列舉資料型態 (Enumeration Data Type)

- 列舉由關鍵字 `enum` 定義，它是一組由識別字所代表的整數列舉常數 (enumeration constant)。
- 除非特別指定，否則 `enum` 內的值都由 0 開始，然後逐漸遞增 1。
- 下方 `enum` 的定義範例，其內部的識別字會分別會設定為整數 0 到 6。

```
1 enum days
2 {
3     SUN, MON, TUE, WED, THU, FRI, SAT
4 };
```

- `enum` 中的識別字如同 `const int` 一樣，是無法修改其值的常數，故通常使用大寫英文來命名。

列舉資料型態 (Enumeration Data Type)

- enum 內的值預設由 0 開始，但也可將 enum 的識別字指定數值，下個識別字的值會是上個識別字遞增 1。
- 下方 enum，其內部的識別字會分別設定為整數 1 到 6 以及 0。

```
1 enum days
2 {
3     MON = 1, TUE, WED, THU, FRI, SAT, SUN = 0
4 };
```

列舉資料型態 (Enumeration Data Type)

- enum 的識別字常被當作符號使用，有時甚至不會在意其內部數值。
- 例如設計遊戲時，可使用以下列舉表達當前狀態：

```
1 typedef enum
2 {
3     CONTINUE, WIN, LOSE
4 } Status;
5
6 int main()
7 {
8     Status status = CONTINUE;
9     while (status == CONTINUE)
10    {
11        // PLAYING GAME
12        // ...
13    }
14    if (status == WIN) printf("You win!\n");
15    else printf("You lose!\n");
16 }
```

附錄：Base64

- ① Base64 介紹
- ② 轉換規則
- ③ 練習題

Base64 介紹

- Base64 是一種基於 64 個可列印字元來表示二進位資料的表示方法。常用於在通常處理文字資料的場合，表示、傳輸、儲存一些二進位資料。
- Base64 常用於電子郵件中二進位資料的傳輸，或是於網頁的文本檔案表示圖片時使用，將圖片資料直接寫在文本檔案中，以減少 client 端請求資料的次數。

轉換規則

- Base64 使用 0-9、A-Z、a-z、+、/ 共 64 個可視字元來表示二進位資料，因為 $\log_2 64 = 6$ ，每個字元可以表示 6 個位元。
- Base64 以 4 個字元為一組，來表示二進位資料中的 3 個位元組。因此，轉換成 Base64 之後，資料長度會比原本多出 33% 以上。
- 例如 I see, I come, I conquer. (共 25 個位元組)，轉換為 Base64 為 SSBzZWUslEkgY29tZSwgSSBjb25xdWVyLg== (共 36 個字元)

Base64 字母表

Index	Binary	Char	Index	Binary	Char	Index	Binary	Char	Index	Binary	Char
0	000000	A	16	010000	Q	32	100000	g	48	110000	w
1	000001	B	17	010001	R	33	100001	h	49	110001	x
2	000010	C	18	010010	S	34	100010	i	50	110010	y
3	000011	D	19	010011	T	35	100011	j	51	110011	z
4	000100	E	20	010100	U	36	100100	k	52	110100	0
5	000101	F	21	010101	V	37	100101	l	53	110101	1
6	000110	G	22	010110	W	38	100110	m	54	110110	2
7	000111	H	23	010111	X	39	100111	n	55	110111	3
8	001000	I	24	011000	Y	40	101000	o	56	111000	4
9	001001	J	25	011001	Z	41	101001	p	57	111001	5
10	001010	K	26	011010	a	42	101010	q	58	111010	6
11	001011	L	27	011011	b	43	101011	r	59	111011	7
12	001100	M	28	011100	c	44	101100	s	60	111100	8
13	001101	N	29	011101	d	45	101101	t	61	111101	9
14	001110	O	30	011110	e	46	101110	u	62	111110	+
15	001111	P	31	011111	f	47	101111	v	63	111111	/
									Padding	=	

轉換規則

- 將二進位資料轉換為 Base64 時，會以 3 個字元組為一組進行轉換。為了方便說明，這裡使用字串作為二進位資料。
- 經過下表所示的轉換，字串 Man 的 Base64 為 TWFu。

Source	Character	M								a								n							
ASCII text	Octets	77 (0x4d)								97 (0x61)								110 (0x6e)							
Bits		0	1	0	0	1	1	0	1	0	1	1	0	0	0	0	1	0	1	1	0	1	1	1	0
Base64 encoded	Sextets	19								22								5							
	Character	T								W								F							
	Octets	84 (0x54)								87 (0x57)								70 (0x46)							

轉換規則

- 如果要編碼的位元組數不能被 3 整除，需先使用 0 位元組值在末尾補足，並加上一個或兩個 = 號，代表補足的位元組數。

Source ASCII text	Character	M								a															
	Octets	77 (0x4d)								97 (0x61)															
Bits		0	1	0	0	1	1	0	1	0	1	1	0	0	0	0	1	0	0						
Base64 encoded	Sextets	19								22								4				Padding			
	Character	T								W								E				=			
	Octets	84 (0x54)								87 (0x57)								69 (0x45)				61 (0x3D)			

Source ASCII text	Character	M																							
	Octets	77 (0x4d)																							
Bits		0	1	0	0	1	1	0	1	0	0	0	0												
Base64 encoded	Sextets	19								16								Padding				Padding			
	Character	T								Q								=				=			
	Octets	84 (0x54)								81 (0x51)								61 (0x3D)				61 (0x3D)			

請參考 Base64 轉換規則的說明，將輸入的字串轉換為 Base64 輸出。

- 範例輸入 1：CE/EE CCU
- 範例輸出 1：Q0UvRUUgQ0NV
- 範例輸入 2：I see, I come, I conquer.
- 範例輸出 2：SSBzZWUslIEkgY29tZSswgSSBjb25xdWVyLg==

Q & A