

# 程式設計 (一)

## CH9. 結構與位元運算

Ming-Hung Wang 王銘宏

tonymhwang@cs.ccu.edu.tw

Department of Computer Science and Information Engineering  
National Chung Cheng University

Fall Semester, 2022

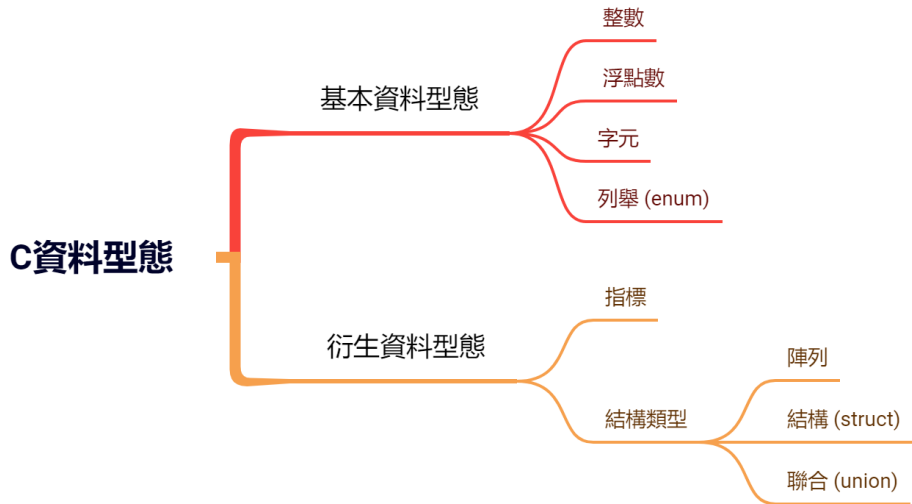
# 目錄

1. 衍生資料型態
2. typedef 宣告
3. 結構的定義與宣告
4. 結構的運算與成員的存取
5. 結構的對齊與位元欄位
6. Union 資料型態
7. 位元運算
8. 列舉資料型態

# 衍生資料型態

## Derived Data Type

# 衍生資料型態



衍生資料型態是由其他型態為基礎來建構的，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態的指標與陣列。

而本章節要介紹的衍生資料型態——結構 (struct)、聯合 (union) 可由複數個不同的資料型態組成。在章節的最後則會介紹類似於整數常數的列舉 (enum) 型態。

# typedef 宣告

## typedef Declarations

# typedef 宣告

typedef 宣告是將識別字宣告為新的資料型態的宣告式。使用 typedef 宣告可以將既有的資料型態取“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

建立 long long 的別名並宣告一個變數：

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

建立 char 相關的一系列別名：

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



# typedef 宣告

使用 typedef 可以將名稱很長的變數型態進行簡化，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態名稱。

# 結構的定義與宣告

## Structure Definition and Declaration

# 結構的定義與宣告

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

```
struct student
{
    char *name;
    int grade;
};
```

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

注意結構的成員不可以包含與自己相同的型態或陣列：

```
struct student
{
    char *name;
    int grade;
    struct student bestFriend; // ERROR
};
```

# 結構的定義與宣告

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**：

```
struct student
{
    char *name;
    int grade;
    struct student *bestFriend; // OK
};
```

## 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：

```
struct student
{
    char *name;
    int grade;
} stdntA, stdntLst[10];
```

```
struct student
{
    char *name;
    int grade;
};
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。

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

# 結構的定義與宣告

## 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：

```
typedef struct student
{
    char *name;
    int grade;
} Student;
Student foo; //宣告結構變數 foo
```

```
struct student
{
    char *name;
    int grade;
};
typedef struct student Student;
Student foo; //宣告結構變數 foo
```



# 結構的定義與宣告

於結構定義時一並宣告別名，其結構標籤 (tag) 可以被省略。

```
typedef struct
{
    char *name;
    int grade;
} Student;

Student foo; //宣告結構變數 foo
```

# 結構的運算與成員的存取

## Structure Member Access

# 結構的運算與成員的存取

## 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。

```
typedef struct {  
    char name[30];  
    int grade;  
} Student;  
  
int main() {  
    Student foo; //宣告結構變數 foo  
    scanf("%s%d", foo.name, &foo.grade);  
    printf("%s grade:%d\n", foo.name, foo.grade);  
}
```

# 結構的運算與成員的存取

## 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`。

```
typedef struct {  
    char name[30];  
    int grade;  
} Student;  
  
void inputStudent(Student *s) {  
    scanf("%s%d", s->name, &s->grade);  
}
```

# 結構的運算與成員的存取

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

結構變數可接受的運算：

- 同樣型態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

在計算結構大小時，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  {
6      char c, d;
7      int i;
8  } Foo;
9
10 int main()
11 {
12     printf("size of Foo: %d\n", sizeof(Foo));
13 }
```

c:\workspace\struct size.exe

size of Foo: 8



# 結構的對齊與位元欄位

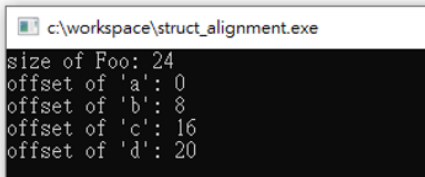
結構的對齊規則 (alignment) 包含以下幾點：

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，型態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

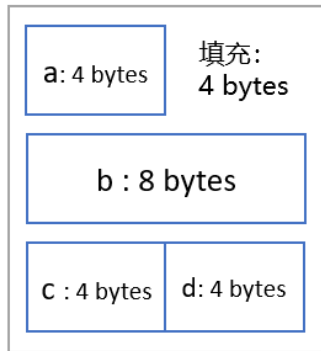
每個成員的偏移量 (offset) 必為該成員大小的倍數。

```
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:\workspace\struct\_alignment.exe

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

結構的大小 (size of) 必為結構成員中，型態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 }
```

c:\workspace\struct\_alignment.exe

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

a: 8 bytes

b : 8 bytes

C : 4 bytes

填充:  
4 bytes

## 位元欄位 (bit fields)

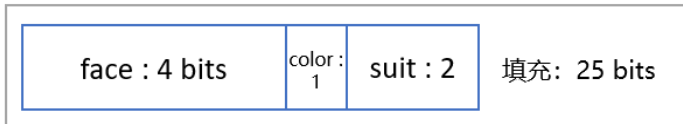
C 語言的結構中，我們可以指定 unsigned int 或 int 成員所佔的位元數量，將資料存放在最少的位元裡，以提高記憶體的使用率。

# 結構的對齊與位元欄位

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。

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



# 結構的對齊與位元欄位

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

使用不具名的欄位，可以填補位元 (padding)，調整成員的偏移量。

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



# 結構的對齊與位元欄位

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)。

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





# Union 資料型態

## Union Data Type

# Union 資料型態

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。

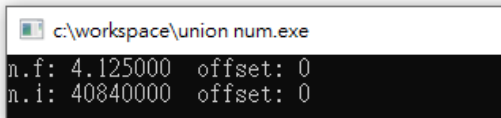
以下是 union 的定義範例。

```
union number
{
    float f;
    int i;
};
```

# Union 資料型態

union 的成員會共用相同的空間，並依照該成員的型態解讀所佔空間中的內容。

```
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 = 4.125;
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 }
```



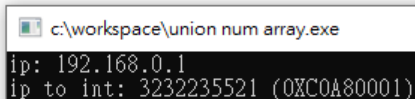
c:\workspace\union num.exe

```
n.f: 4.125000  offset: 0
n.i: 40840000  offset: 0
```

# Union 資料型態

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", ip.bytes[3], ip.bytes[2], ip.bytes[1], ip.bytes[0]);
14     printf("ip to int: %u (%#X)\n", ip.i, ip.i);
15 }
```



c:\workspace\union num array.exe  
ip: 192.168.0.1  
ip to int: 3232235521 (0XC0A80001)

# 位元運算

## Bitwise Operation

# 位元運算

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

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	$\wedge$	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

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  $\sim x$ ,

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

## 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("Input 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 }
```

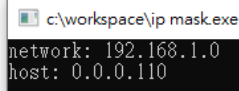
c:\workspace\unsigned to bin.exe

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

# 位元運算

## 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]=1, [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 }
```



c:\workspace\ip mask.exe  
network: 192.168.1.0  
host: 0.0.0.110

# 列舉資料型態

## Enumeration Data Type

# 列舉資料型態

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。

```
enum days
{
    SUN, MON, TUE, WED, THU, FRI, SET
};
```

`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

enum 內的值預設由 0 開始，但也可將 enum 的識別字指定數值，下一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。

```
enum days
{
    MON=1, TUE, WED, THU, FRI, SET ,SUN=0
};
```

# 列舉資料型態

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：

```
typedef enum
{
    CONTINUE, WIN, LOSE
} Status;

int main()
{
    Status status = CONTINUE;
    while (status == CONTINUE)
    {
        // PLAYING GAME
        //...
    }
    if (status == WIN) /*SHOW USER WIN*/ ;
    else (status == LOSE) /*SHOW USER LOSE*/ ;
}
```