

第十一章

結構 (Structure)

大綱

結構宣告與初值設定

結構與陣列

結構與指標

結構與函數

列舉型態

自訂型態

認識結構 — 結構定義

- 結構(struct)可將資料組合成為一個新的**型態**
- 定義結構

```
struct 結構名稱
{
    資料型態 成員名稱1;
    資料型態 成員名稱2;
    ...
    資料型態 成員名稱n;
};
```

定義成員

定義結構

- 結構定義範例

```
struct st_date
{
    int yy;
    int mm;
    int dd;
};
```

認識結構 — 結構變數


■ 宣告結構變數

```
struct 結構名稱 結構變數1, 結構變數2;
```

```
struct st_date date1, date2;
```

■ 存取結構變數的成員

結構變數名稱.成員名稱;



結構成員存取運算子

```
date1.yy = 1982;
```

範例 – 結構

- 定義st_stu結構，包含id及name[40]成員

```
struct st_stu          //定義結構
{
    int id;
    char name[40];
};

int main(void)
{
    struct st_stu stu1,stu2;    //宣告結構變數

    printf("Enter Student ID: ");
    scanf("%d", &stu1.id);      //取記憶體位址

    printf("Your Student ID: %d", stu1.id);
}
```

結構變數 – 初值設定

- 結構變數的初值比照陣列初值設定方式
- 指定運算元(=)可使用於結構

```
struct st_stu stu1 = {2008, "Barack Obama"};
struct st_stu stu2;

printf("Student ID: %d\n", stu1.id);
printf("Student Name: %s\n", stu1.name);

stu2 = stu1;    //設定stu2內容與stu1相同
printf("Student ID: %d\n", stu2.id);
```


巢狀結構

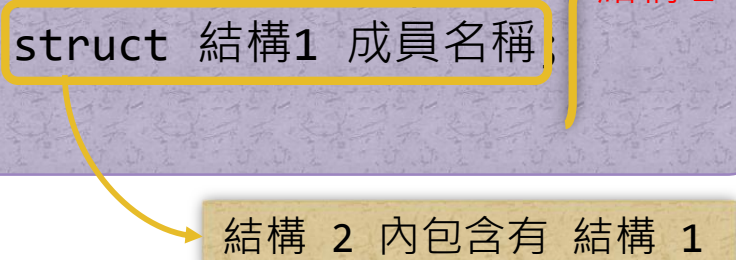
- 結構內可包含其他結構
- 巢狀結構：結構內包含其他結構

```
struct 結構1
{
    /* 結構1的成員 */
};

struct 結構2
{
    /* 結構2的成員 */
    struct 結構1 成員名稱;
};
```

結構 1

結構 2



結構 2 內包含有 結構 1

```
struct st_birth
{
    int year;
    int month;
    int day;
};

struct st_stu
{
    int id;
    char name[40];
    struct st_birth birth;
};
```

範例 – 巢狀結構

■ 巢狀結構給定初值比照多維陣列

```
int main(void)
{
    struct st_stu stu1
        = {2008, "Barack Obama", {1961, 8, 4}};

    printf("%d %s: %d/%d/%d\n",
           stu1.id,
           stu1.name,
           stu1.birth.year,
           stu1.birth.month,
           stu1.birth.day);
}
```


結構陣列

■ 結構變數比照一般變數宣告陣列

`struct 結構型態 結構陣列名稱[元素個數];`

```
struct st_stu stu[2] = {  
    {2008, "Barack Obama", {1961, 8, 4}},  
    {2004, "George Bush", {1947, 6, 6}}  
};
```

```
for (i=0; i<2; i++)  
    printf("%d %s: %d/%d/%d\n",  
        stu[i].id,  
        stu[i].name,  
        stu[i].birth.year,  
        stu[i].birth.month,  
        stu[i].birth.day);
```

結構指標

■ 結構指標變數的宣告

```
struct 結構型態 *結構指標變數名稱;
```

```
struct st_date *p;
```

■ 結構指標的設定

```
結構指標變數 = &結構變數;
```

```
p = &date1;
```

■ 運用結構指標存取結構成員

```
結構指標變數->結構成員
```

```
(*結構指標變數).結構成員
```

```
p->day = 12;  
(*p).day = 12;
```

相同意義

範例 – 結構指標

■ 指標運用 -> 運算子存取結構成員

```
struct st_stu stu[2] = {  
    {0, "Barack Obama", {1961, 8, 4}},  
    {2004, "George Bush"}  
};  
  
int y1 = 2008;  
struct st_birth day1 = {1947, 6, 6};  
  
struct st_stu *p1 = stu, *p2; //stu為陣列(靜態指標)  
struct st_birth *p3 = &day1; //day1為結構  
  
(*p1).id = y1; //相當於 p1->id = y1;  
p1++;  
p2 = p1;  
  
p2->birth = *p3; //p2->birth為st_birth類型
```

函數 — 傳遞結構及指標

■ 將結構傳遞到函數

```
傳返回值型態 函數名稱(struct 結構名稱 變數名稱)
{
    /* 函數的定義 */
}
```

```
void show_stu(struct st_stu stu1);
```

■ 將結構指標傳遞到函數

```
傳返回值型態 函數名稱(struct 結構名稱 *指標變數名稱)
{
    /* 函數的定義 */
}
```

```
void print_stu(struct st_stu *p);
```

範例 – 函數傳遞非結構

■ 函數傳遞陣列元素

```
int main(void)
{
    int id[2] = {2008,2004};
    char name[][20] = {"Barack Obama", "George Bush"};
    int yy[] = {1961,1947};
    int mm[] = {8,6};
    int dd[] = {4,6};
    int i;
    for (i=0;i<2;i++)
        show_stu(id[i], name[i], yy[i], mm[i], dd[i]);
}

void show_stu(int id, char n[], int yy, int mm, int dd)
{
    printf("%d:%d\n", id, yy);
}
```

範例 – 函數傳遞結構

■ 函數傳遞陣列元素

```
int main(void)
{
    struct st_stu stu[2] = {
        {2008, "Barack Obama", {1961, 8, 4}},
        {2004, "George Bush", {1947, 6, 6}}
    };

    int i;
    for (i=0; i<2; i++)
        show_stu(stu[i]);    //傳遞結構指標
}

void show_stu(struct st_stu s)
{
    printf("%d:%d\n", s.id, s.birth.year);
}
```

範例 – 函數傳遞結構位址

■ 函數傳遞陣列元素位址

```
int main(void)
{
    struct st_stu stu[2] = {
        {2008, "Barack Obama", {1961, 8, 4}},
        {2004, "George Bush", {1947, 6, 6}}
    };
    struct st_stu *p = stu;
    int i = 0;
    for (i=0; i<2; i++)
        print_stu(p++);           //傳遞結構指標
}

void print_stu(struct st_stu *p)
{
    printf("%d:%d\n", p->id, p->birth.year);
}
```


範例 – 函數傳遞結構陣列

■ 函數列印結構陣列

```
int main(void)
{
    struct st_stu stu[2] = {
        {2008, "Barack Obama", {1961, 8, 4}},
        {2004, "George Bush", {1947, 6, 6}}
    };
    print_stu_ary(stu);
}

void print_stu_ary(struct st_stu *p)
{
    struct st_stu *q;
    for (q=p; q-p<2; q++)
        printf("%d:%d/\n", q->id, q->birth.year);
}
```

列舉型態 (enum)

- 列舉型態 (enumeration)
 - 可以用有意義的名稱來取代整數常數
- 列舉型態定義

```
enum 列舉名稱  
{  
    列舉常數1,  
    列舉常數2,  
    ...  
    列舉常數n  
};
```

- 列舉型態變數的宣告

```
enum 列舉名稱 變數1, 變數2,..., 變數m;
```

```
enum en_color  
{  
    green,    // 0  
    yellow,  // 1  
    red       // 2  
};  
  
enum en_model  
{  
    first,      // 0  
    five = 5,   // 5  
    six,        // 6  
};
```

範例 – 列舉型態

- 使用green/yellow/red替代0/1/2

```
int main(void)
{
    enum en_color light = green;

    if (light == green)
        printf("Go!\n");
    else if (light == yellow)
        printf("Wait!");
    else if (light == red)
        printf("Stop!\n");
}
```

自訂型態 (typedef)

- typedef 可定義新的資料型名稱
 - 方便程式撰寫
 - 方便程式閱讀

```
typedef 資料型態 識別字;
```

```
struct st_date  
{  
    int year;  
    int month;  
    int day;  
};  
typedef struct st_date DATE;
```



```
typedef struct  
{  
    int year;  
    int month;  
    int day;  
} DATE;
```

範例 – typedef

- 定義DATE結構類型
- 定義BOOL列舉類型

```
typedef struct
{
    int year;
    int month;
    int day;
} DATE;

typedef enum
{
    false,
    true
} BOOL;
```

```
int main(void)
{
    DATE d1 = {2008,3,28};
    BOOL b1 = true;

    show_day(d1);

    printf("The answer is ");
    if (b1==true)
        printf("true.\n");
    else
        printf("false!\n");
}

void show_day(DATE d)
{
    printf("%d/%d/%d\n",
        d.year,d.month,d.day);
}
```

Any question?

