程式設計
(Programming)

真理大學 資訊工程系 吳汶涓老師

CH10
結構、Unions、位元處理以及列舉型別

本章綱要

10.1 簡介

- 結構
  - 將一些彼此相關的變數結合成相同的名稱，有時也可稱爲聚合體 **(aggregate)**
    - 可以含有不同資料型別的變數
  - 常會用來定義儲存在檔案裡的記錄
  - 指標和結構可以構成複雜的資料結構，例如鏈結串列、佇列、堆疊和樹(tree)

name	age	gender	hourSalary
John	21	M	120
May	25	F	300
Sue	33	F	250

10.2 結構定義

- 範例

```
struct student{
    char name[10];
    int age;
};
```

name	age
John	21

常見的程式設計錯誤 10.1

忘了用分號來結束結構的定義。此爲語法錯誤

- struct 開始student結構的定義
- **student是結構名稱**，可用來宣告該結構型別的變數
- student包含兩個成員，一個型別爲字元陣列、一個型別爲整數
  - 它們是name 和age

- 類似陣列，但結構可含有許多不同的資料型態的變數

```
struct employee{
    char name[20];
    int age;
    char gender;
    double hourSalary;
};
```

name	age	gender	hourSalary
John	21	M	120

- struct的資訊
  - 結構不能包含它自己
  - 可以包含指向相同結構型別的指標成員 (稱爲巢狀結構)
  - 結構的定義並沒有佔用任何的記憶體空間
    - 它建立一種新的資料型別，以供結構變數宣告之用

```
struct employee{
        char name[20];
        int age;
        char gender;
        double hourSalary;
        struct employee person;
        struct employee *ePtr;
};
```

此行錯誤

10.3 結構的宣告與初始值設定

- 結構的宣告
  - 與其他型別的宣告相同
    struct employee P1, person[50], *p1Ptr;
  - 或是，宣告與結構定義一起寫

```
struct employee{
        char name[20];
        int age;
        char gender;
        double hourSalary;
} P1, person[50], *p1Ptr;
```

- 結構的初始值設定
  struct employee P1={“John”, 21, ‘M’, 120};

良好的程式設計習慣 10.1

當你在建立結構型別時，請務必爲它提供一個結構標籤名稱。如此你在往後的程式裡才能夠方便宣告這種結構型別的變數。

如同先前的**employee**

良好的程式設計習慣 10.2

選擇具有意義的結構標籤名稱，將有助於程式的可讀性

- 結構不能用==和!=運算子來進行比較，因爲結構的成員並不一定會存放在連續記憶體空間

```
struct employee P1={“John”, 21, ‘M’, 120};
struct employee P2={“May”, 25, ‘F’, 300};
if(P1==P2) //此行錯誤
```

10.4 存取結構

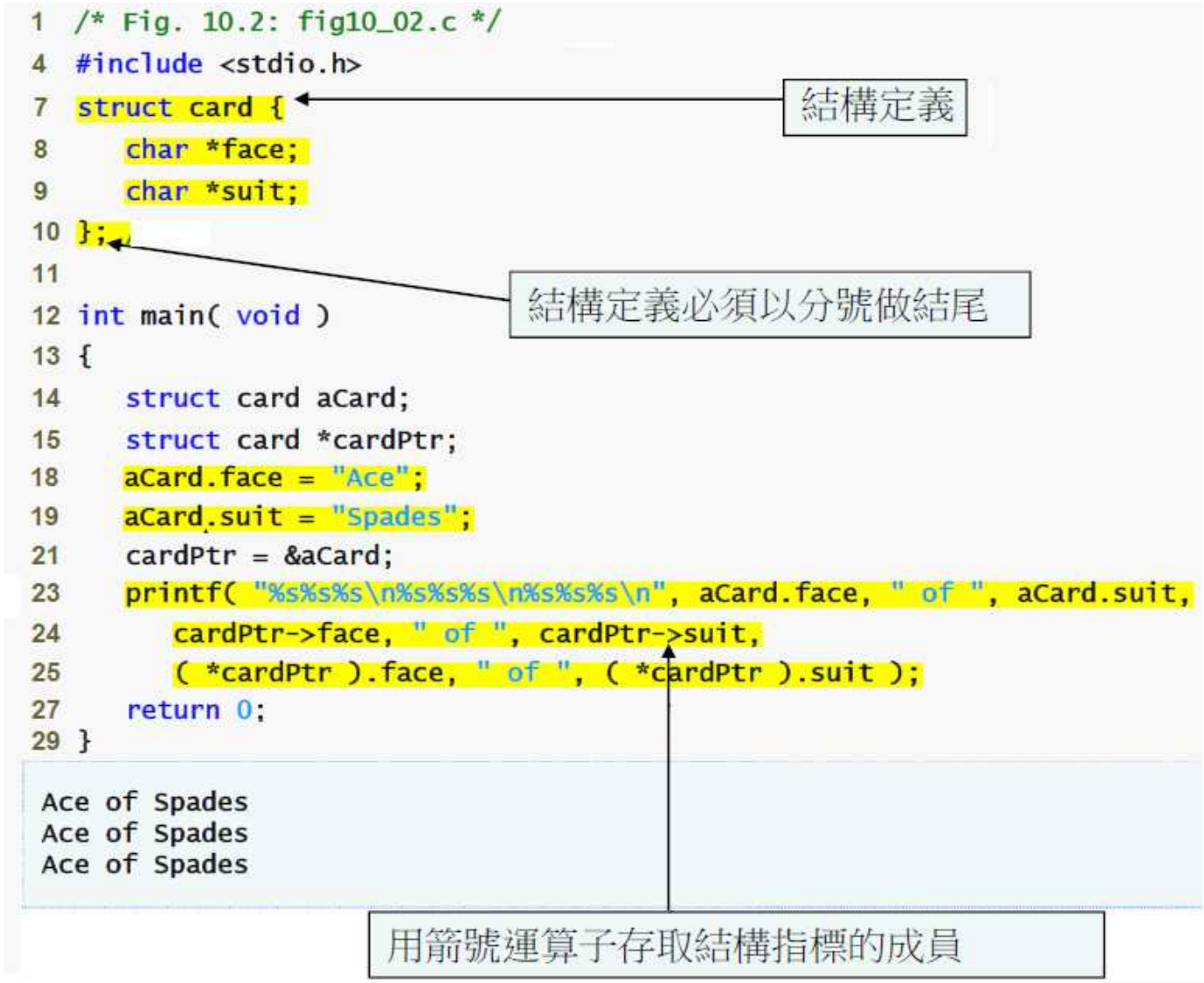
- 存取結構成員
  - 使用結構變數和點號運算子**(.)**

```
struct employee P1;
P1.name="John";
printf("%s", P1.name);
```

  - 使用指向結構變數的指標和箭號運算子**(->)**

```
struct employee *p1Ptr = &P1;
p1Ptr->name="John";
printf("%s", p1Ptr->name);
```

  - **p1Ptr->name** 等於 **(*p1Ptr).name**


/* Fig. 10.2: fig10_02.c */
#include <stdio.h>
struct card {
char *face;
char *suit;
};
int main(void)
{
struct card aCard;
struct card *cardPtr;
aCard.face = "Ace";
aCard.suit = "Spades";
cardPtr = &aCard;
printf("%s%s%s\n%s%s%s\n%s%s%s\n", aCard.face, " of ", aCard.suit,
cardPtr->face, " of ", cardPtr->suit,
(*cardPtr).face, " of ", (*cardPtr).suit);
return 0;
}
Ace of Spades
Ace of Spades
Ace of Spades
結構定義
結構定義必須以分號做結尾
用箭號運算子存取結構指標的成員

良好的程式設計習慣 10.3

請勿在 -> 和運算子的兩邊留下任何的空白。如此可以突顯包含此運算子的運算式爲單一的變數名稱。

常見的程式設計錯誤 10.4

試圖只用成員名稱來參考結構的成員。此爲語法錯誤。

常見的程式設計錯誤 10.5

使用指標和結構成員運算子參考結構的成員時沒有加上括號 (例如 `*cardPtr.suit`) 是一個語法錯誤。

- 避免對不同型別的結構成員使用相同的名稱。雖然這麼做是合法的，但很容易產生混淆。

練習

- 使用結構撰寫程式，假設班上僅有5 個人，求數學、英文與程設的平均分數。請使用結構型態宣告一個5個學生的陣列，如struct student person[5];

David: 74, 80, 66
John: 72, 90, 77
Sue: 77, 65, 60
Sally: 65, 58, 74
Kevin: 81, 79, 68

10.5 使用結構與函式

- 將結構傳遞給函式
  - 傳遞整個結構　或　傳遞個別的結構成員
  - 都是傳值呼叫
- 以傳參考呼叫的方式傳遞結構
  - 傳遞它的位址
  - 傳遞指向它的參照

- 範例：(傳值呼叫)

```
struct employee{
    char name[20];
    int age;
    char gender;
    double hourSalary;
};
```

```
void display(struct employ T){
    printf("%s\n", T.name);
    printf("%d\n", T.age);
    printf("%c\n", T.gender);
}
```

```
int main(void){
    struct employee P1;
    display(P1);
    return 0;
}
```

- 範例：(傳參考呼叫)

```
struct employee{
      char name[20];
      int age;
      char gender;
      double hourSalary;
};
```

```
void display(struct employ *ptr){
      printf("%s\n", ptr->name);
      printf("%d\n", ptr->age);
      printf("%c\n", ptr->gender);
}
```

```
int main(void){
      struct employee P1;
      display(&P1);
      return 0;
}
```

增進效能的小技巧 10.1

以傳參考呼叫來傳遞結構會比傳值呼叫更有效率 (傳值呼叫會將整個結構複製一份)。

練習

- (續前練習)使用結構撰寫程式，假設班上僅有5 個人，求數學、英文與程設的平均分數。請將計算平均分數的程式碼改寫成函式，以傳參考方式來傳遞結構。

  David: 74, 80, 66
  John: 72, 90, 77
  Sue: 77, 65, 60
  Sally: 65, 58, 74
  Kevin: 81, 79, 68

練習

- 使用結構來建立班上同學基本資料，班上同學數量n由使用者輸入，需記錄同學的姓名、電話、年齡、住址等基本資料。輸入完畢後，利用傳參考呼叫方式將所有同學資料輸出於螢幕上。如：
  “David”, 982117488, 23, “新北市淡水區真理街32號”
  “Sue”, 972117883, 21, “新北市新店區中正路100號”