【第九章】

結構

講師: 李根逸 (Ken-Yi Lee), E-mail: feis.tw@gmail.com

課程大綱

- 定義結構 (struct) [P243]
- 結構變數宣告 [P244]
  - 結構變數宣告的初始化 [P246]
- 存取結構成員 [P247]
- 傳送大型資料型態參數 [P251]
- 列舉型態 (enum) [P252]

定義結構

- 結構是一種衍生的資料型態，可以由一到多個資料型態的成員所建構的
- 基本定義語法：

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

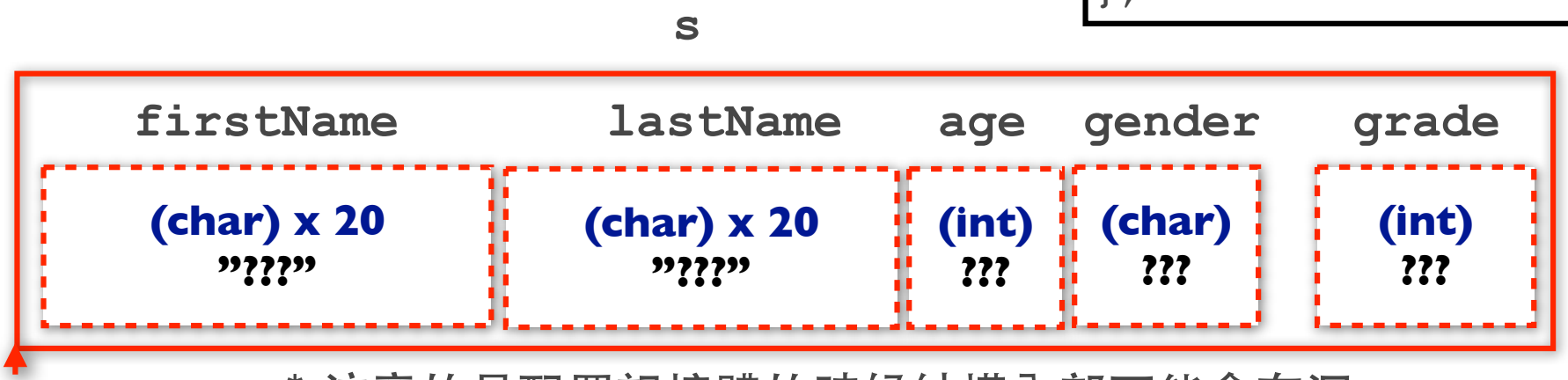
- 範例：

```
struct Student {
     char firstName[20];
     char lastName[20];
     int age;
     char gender;
     int grade;
};
```

結構變數宣告

- 定義好結構後，就可以就可以宣告以結構為型別的變數了：
  - struct 結構名稱 變數名稱; /* C 語法 */
  - 結構名稱 變數名稱; /* C++ 語法 */
- 範例： Student s;
  - 結構變數會佔據一段連續的記憶體

```
struct Student {
    char firstName[20];
    char lastName[20];
    int age;
    char gender;
    int grade;
};
```



22936**20**

* 注意的是配置記憶體的時候結構內部可能會有洞

《補充》typedef

- 在 **C** 裡面, 要宣告一個 **struct** 的變數需要寫成：
  - struct 結構名稱 變數名稱;
    - 例子：struct Student s;
- 因此需要用 **typedef** 來幫 **struct** 取型別別名：
  - `typedef struct` 結構名稱 型別名稱;
    - 例子：typedef struct Student StudentType;
- 之後才能用型別名稱直接宣告變數：
  - 型別名稱 變數名稱;
    - 例子：StudentType s;
- 在 C++ 中則可以直接用結構名稱當型別宣告變數

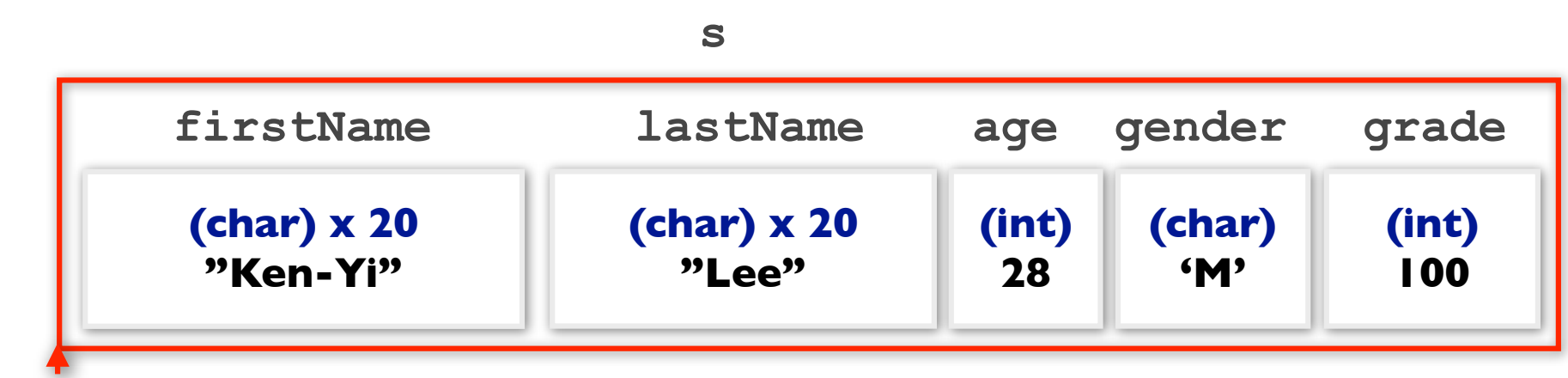
結構變數宣告的初始化

- 結構變數可以使用如同陣列般的初始化方式：

  結構名稱 變數名稱 = { 第一個成員的值, 第二個成員的值, ... }

- 範例：

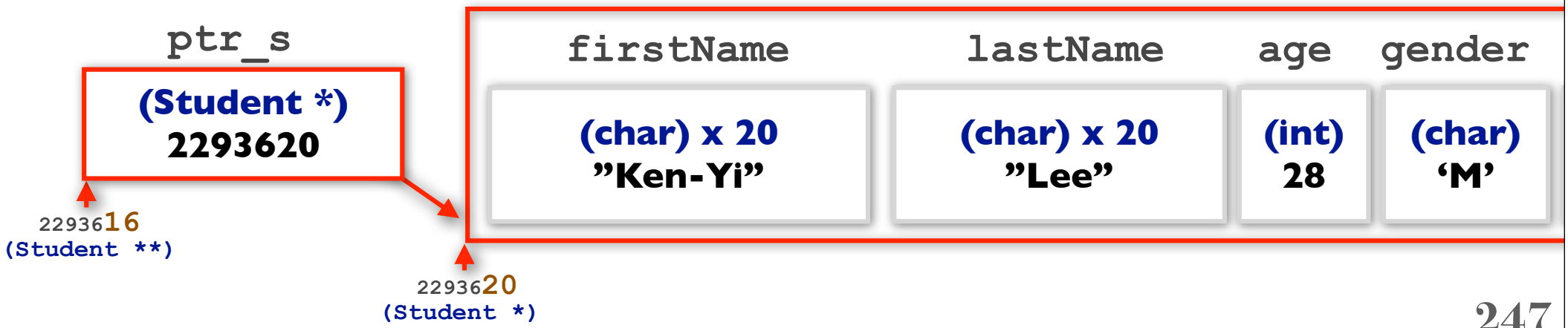
```
Student s = { "Ken-Yi", "Lee", 28, 'M', 100 };
```



存取結構成員

- 有兩種運算子可以存取結構成員：
  - 結構成員運算子 [點號] (.)：對一般結構變數使用
    - 範例:
      - s.firstName
  - 結構指標運算子 [箭號] (->)：對結構指標變數使用
    - 範例:
      - Student *ptr_s = &s;
      - ptr_s->firstName

-> 和 . 的優先順序比 * 高



《範例》資料庫

- 試寫一程式 **(db.cpp)**，讓使用者輸入三位學生的資料後，讓使用者再輸入一個 **1 ~ 3** 的數字後，顯示該編號的學生資料

```
struct Student {
    char firstName[20];
    char lastName[20];
    int age;
    char gender;
    int grade;
};
```

  - 一般使用結構的好處:
    - 簡化程式碼, 讓程式更容易看懂 (資料包裝)
    - 讓程式碼更有彈性 (資料封藏)

傳值還是傳址？

- 在設計函式時，在 C 裡面可以選擇直接傳值或傳變數位址:
  - 傳值: `void print1(int num);`
  - 傳址: `void print2(int *num);`

送入函式	傳值	傳址
能改變原本變數的值?	不能	可能
需複製的記憶體大小	依資料型態	固定 (例: 32-bit)

- 傳值可以保護原變數的值不被修改，但是在傳結構或陣列這類大型資料型態時可能會比較花時間

再論 const 修飾字

- 當我們想用傳位址的方式傳送一個變數給其他函式，但是又不希望該函式能修改該變數的值，我們可以在宣告函式引數時，加上 **const** 修飾字：
  - `void print3(const int *num);`
    - 只是這個例子在效率上跟 `void print1(int num);` 比起來沒有優勢，因為 `int` 是個小型資料型態。

傳送大型資料型態參數

- 結構或陣列因為通常會佔據比較大的記憶體空間，所以傳送到函式內的時候我們會傾向用傳址的方式。但是我們又同時希望原變數的值可以受到保護，不會被呼叫的函式內部改變。此時我們可以利用 **const** 修飾字:
  - 結構的例子:
    - `void printBad1(Student s);  // 傳值 -> 慢`
    - `void printBad2(Student* s);  // 傳址 -> 不安全`
    - `void printGood(const Student* s);`
  - 陣列的例子:
    - `void printBad1(Student s[3]); // 傳址 -> 不安全`
    - `void printBad2(Student *s);  // 傳址 -> 不安全`
    - `void printGood(const Student s[3]);`

列舉型態 (enum)

- 列舉型態的語法:

```
enum 列舉型態名稱 {
   列舉常數名稱1,
   列舉常數名稱2,
   列舉常數名稱3,
   ...
};
```

參考 enum.cpp

習題 (1)

- **[E0901] 試著撰寫一名為 time 的結構，包含 hour, min 跟 sec 等三個成員 (型態都為 int)**
  - 撰寫一函式 `void read(time *t);` 讓使用者輸入目前時間
  - 撰寫一函式 `void step(time *t);` 讓 t 的時間增加一秒
  - 撰寫一函式 `void print(const time *t);` 印出目前時間 (目前時間為 XX 點 XX 分 XX 秒！)
  - 撰寫一函式 `time interval(const time *t1, const time *t2);` 傳回 t1 與 t2 的時間差距