

# HW5 期中考重做

班級	姓名	學號	日期
四機械四乙	吳宇昕	B10831020	11/20/2022

## 簡答題

### 1. 請說明getter與setter之意義

class內變數的accessibility預設為private，不可被class外的程式讀寫，達到information hiding的作用。

然而若有需求，必須在class外（例如main函式裡）讀寫class member的值，與其直接把該class member移至public區域，失去information hiding的設計，我們可以設定一個在class內的public函式，專門讀寫特定class member的值。讀取member值的函式稱為getter，而寫入member值的函式稱為setter。

如此一來，class member既沒有失去information hiding，又可以在需要的時候被class外的程式讀寫。

### 2. 參數傳遞概念

函式參數可以選擇傳入引數的值，或傳入引數的reference，即記憶體位置。若函式只需要接收參數的數值進行運算，可以將引數值用pass by value的參數傳入函式。函式內的參數為引數值的副本，因此修改該參數不會影響在函式外被當作引數傳入的變數。pass by value用在函式不希望修改函式外的變數的場合

pass by reference會傳進某個函式外變數的記憶體位置(reference)，而不是複製其值給函式內的參數產生副本。若函式需要輸出多個計算結果，可以給它pass by reference的參數，將計算結果寫入函式外的某個變數的記憶體位置。

由於pass by value需要複製變數值，若參數是個龐大的物件，過度使用pass by value會使程式效能較差。此時就可以考慮用pass by reference傳入引數的記憶體位置，避免複製值。若想必免在函式裡修改該參數，則可以使用pass by const reference。

## 第1A階段

[source code](#) and [replit](#)

終端機輸出:

Name	Eric	ID	0
salary	1500		
hour rate	100		
Hours	15		
-----			
Name	NotEric	ID	1
salary	1980		
hour rate	110		
Hours	18		
-----			
Name	NotNotEric	ID	2
salary	2400		
hour rate	120		
Hours	20		
-----			
Total salary:	5880		

**Hours** 的 **setter**定義如下:

```

15     void SetHours(int hours)
16     {
17         if(hours <= 50 and hours > 0)
18             this->Hours = hours;
19         else
20             std::cout << "Hour error" << std::endl;
21         this->Salary = this->HourRate * this->Hours;
22     }

```

`PartTimeWorker::SetHours()` 函式除了會設定各個員工的時數，還會順便把他的薪水一起算好存進 `this->Salary`。對一個 `setter` 來說，這應該不是個好作法。`setter` 的功能應該越單純越好，避免日後使用這個 `struct` 的時候，只是想設定時數卻意外修改到員工的薪水，產生很不直觀的 `bug`。但是考試題目後續沒有說我們可以寫另一個計算薪水的函式或 `setter`，而且下一小題就是寫 `Salary` 的 `getter`，只好在這裡就把員工的薪水算出來了。

**Salary** 的 **getter**定義如下:

```
float PartTimeWorker::GetSalary() const{return this->Salary;}
```

原本想要到 `salary` 的 `getter` 才把員工的薪水算好，但是讓 `getter` 修改 `instance` 值似乎更奇怪。所以就把 `getter` 宣告成 `const` 了，單純回傳薪水值。

**main** 函式內容如下:

```

40  int main()
41  {
42      const int MAX_EMPLOYEE_COUNT = 8;
43      std::array<PartTimeWorker, MAX_EMPLOYEE_COUNT> employees;
44      employees[0] = {0, "Eric", 100, 0, 0};
45      employees[0].SetHours(15); // PartTimeWorker::SetHours() also calculates and sets the salary of each instance
46      employees[1] = {1, "NotEric", 110, 0, 0};
47      employees[1].SetHours(18);
48      employees[2] = {2, "NotNotEric", 120, 0, 0};
49      employees[2].SetHours(20);
50
51      float totalSalary = 0;
52      for(int i = 0; i < 3; i++){
53          std::cout << employees[i] << std::endl;
54          totalSalary += employees[i].GetSalary();
55          std::cout << "-----" << std::endl;
56      }
57      std::cout << "Total salary:" << std::setw(8) << std::setprecision(6) << totalSalary << std::endl;
58      system("pause");
59  }

```

宣告了一個常數 `MAX_EMPLOYEE_COUNT = 8`，及員工陣列 `std::array<PartTimeWorker, MAX_EMPLOYEE_COUNT> employees`

感覺這樣寫程式並不理想。由於每個員工的名字不一樣，需要一個個宣告 `PartTimeWorker` instance 並用 `initializer list` 取名，然後呼叫 `SetHours()` 函式。同樣的程式碼需要寫三次，不能用 `for loop`。

## 第1B階段

[source code](#) and [replit](#)

終端機輸出:

```

Name          Eric      ID  0
  salary      1500
hour rate    100
Hours         15
-----
Name          NotEric    ID  1
  salary      1980
hour rate    110
Hours         18
-----
Name          Erica      ID  3
  salary      4692
hour rate    138
Hours         34
-----
Name          NotErica    ID  4
  salary      6016
hour rate    128
Hours         47
-----

```

原本有三個員工，Eric, NotEric與NotNotEric。刪除NotNotEric，並加入兩位新員工，Erica與NotErica

**main** 函式定義如下:

```
36  int main()
37  {
38      const int MAX_EMPLOYEE_COUNT = 8;
39      PartTimeWorker employees[MAX_EMPLOYEE_COUNT];
40      employees[0] = {0, "Eric", 100, 0, 0};
41      employees[0].SetHours(15); // PartTimeWorker::SetHours() also calculates and sets the salary of each instance
42      employees[1] = {1, "NotEric", 110, 0, 0};
43      employees[1].SetHours(18);
44      employees[2] = {2, "NotNotEric", 120, 0, 0};
45      employees[2].SetHours(20);
46
47      // Overwrites employees[2]
48      employees[2] = {3, "Erica", 138, 0, 0};
49      employees[2].SetHours(34);
50
51      // New employees[3]
52      employees[3] = {4, "NotErica", 128, 0, 0};
53      employees[3].SetHours(47);
54
55      for(int i = 0; i < 4; i++){
56          std::cout << employees[i] << std::endl;
57          std::cout << "-----" << std::endl;
58      }
59      system("pause");
60  }
```

題目要求「刪除舊員工一位」，但是陣列似乎沒辦法直接刪除一個元素，長度縮短。因此我的作法是重新宣告陣列中index 2的PartTimeWorker元素，讓新的PartTimeWorker instance覆蓋舊的，看起來像是刪除員工。

其餘的程式碼與1A相同。

## 第2階段

[source code](#) and [replit](#)

終端機輸出:

```

Name      Eric      ID  0
  salary   1500
hour rate  100
Hours      15
-----
Name      NotEric   ID  1
  salary   1980
hour rate  110
Hours      18
-----
Name      Erica     ID  3
  salary   4692
hour rate  138
Hours      34
-----
Name:      Erica     ID  3
  salary   4692
hour rate  138
hours      34
-----
Name:      NotEric   ID  4
  salary   6580
hour rate  140
hours      47
-----
Press any key to continue . . .

```

保持前一題的情形，覆蓋其中一個員工，並且加入兩位新的

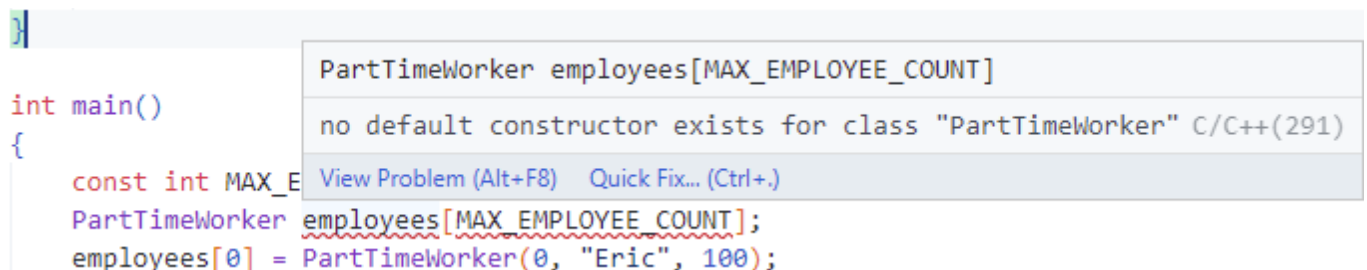
### Constructor 定義:

```

16     PartTimeWorker(int empID, std::string name, float hourRate)
17         :mEmpID(empID), mName(name), mHourRate(hourRate) {}
18
19     PartTimeWorker(){} // dummy constructor

```

有點納悶為什麼會需要19行的空白constructor。若在class裡新增第16行的constructor，便會覆蓋C++ compiler為每個class預設的空白constructor。這時候會出現下面這種compile time error:



```

int main()
{
    const int MAX_E
    PartTimeWorker employees[MAX_EMPLOYEE_COUNT];
    employees[0] = PartTimeWorker(0, "Eric", 100);
}

```

PartTimeWorker employees[MAX\_EMPLOYEE\_COUNT]  
no default constructor exists for class "PartTimeWorker" C/C++(291)  
View Problem (Alt+F8) Quick Fix... (Ctrl+.)

main函式宣告儲存PartTimeWorker的陣列壞掉了？錯誤訊息說

no default constructor exist for class PartTimeWorker

不知道為什麼，好像儲存class instance的陣列宣告當下都很需要空白constructor，沒有空白constructor就不能用。

心得

這份作業最讓我納悶的，是前面提到關於空白constructor的問題。實在想不出來為什麼宣告陣列儲存class instance會需要用到一個空白constructor，特別寫一個空白constructor與真正要用到的自訂constructor並存。

若我的陣列不是儲存class instance，而是儲存instance的pointer，那不用空白constructor就可以正常compile並執行，沒有問題。