

多檔案專案裡的 Circular Relationship

弄不清楚這件事，
你會覺得把程式分成多個檔案時，
編譯器一直找你麻煩，
連編譯都不成功的話，別說測試程式了

出現什麼狀況？

出現什麼狀況？

```
// A.cpp
#include "A.h"
void A::service(B& b) {
    ...
}
```

出現什麼狀況？

```
// A.h
#ifndef _A_H
#define _A_H
#include "B.h"
class A {
public:
    void service(B& b);
private:
    B *m_bptr;
};
#endif
```

```
// A.cpp
#include "A.h"
void A::service(B& b) {
    ...
}
```

出現什麼狀況？

```
// A.h
#ifndef _A_H
#define _A_H
#include "B.h"
class A {
public:
    void service(B& b);
private:
    B *m_bptr;
};
#endif
```

```
// B.h
#ifndef _B_H
#define _B_H
#include "A.h"
class B {
public:
    void service(A a);
private:
    A m_aComponent;
};
#endif
```

```
// A.cpp
#include "A.h"
void A::service(B& b) {
    ...
}
```

出現什麼狀況？

```
// A.h
#ifndef _A_H
#define _A_H
#include "B.h"
class A {
public:
    void service(B& b);
private:
    B *m_bptr;
};
#endif
```

```
// B.h
#ifndef _B_H
#define _B_H
#include "A.h"
class B {
public:
    void service(A a);
private:
    A m_aComponent;
};
#endif
```

undefined

```
// A.cpp
#include "A.h"
void A::service(B& b) {
    ...
}
```

- 編譯 A.cpp 時編譯器說 B.h 裡面 A 是沒有定義過的

出現什麼狀況？

```
// A.cpp  
#include "A.h"  
void A::service(B& b) {  
    ...  
}
```

```
// B.cpp  
#include "B.h"  
void B::service(A a) {  
    ...  
}
```

- 編譯 A.cpp 時編譯器說 B.h 裡面 A 是沒有定義過的

出現什麼狀況？

```
// B.h
#ifndef _B_H
#define _B_H
#include "A.h"
class B {
public:
    void service(A a);
private:
    A m_aComponent;
};
#endif
```

```
// A.cpp
#include "A.h"
void A::service(B& b) {
    ...
}
```

```
// B.cpp
#include "B.h"
void B::service(A a) {
    ...
}
```

- 編譯 A.cpp 時編譯器說 B.h 裡面 A 是沒有定義過的

出現什麼狀況？

```
// A.h
#ifndef _A_H
#define _A_H
#include "B.h"
class A {
public:
    void service(B& b);
private:
    B *m_bptr;
};
#endif
```

```
// B.h
#ifndef _B_H
#define _B_H
#include "A.h"
class B {
public:
    void service(A a);
private:
    A m_aComponent;
};
#endif
```

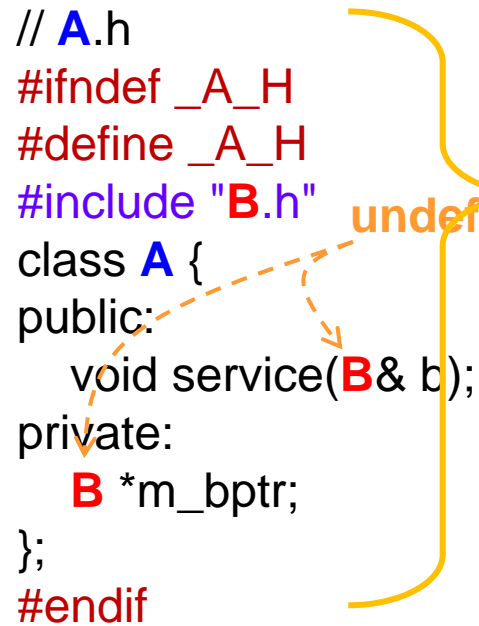
```
// A.cpp
#include "A.h"
void A::service(B& b) {
    ...
}
```

```
// B.cpp
#include "B.h"
void B::service(A a) {
    ...
}
```

- 編譯 A.cpp 時編譯器說 B.h 裡面 A 是沒有定義過的

出現什麼狀況？

```
// A.h
#ifndef _A_H
#define _A_H
#include "B.h"
class A {
public:
    void service(B& b);
private:
    B *m_bptr;
};
#endif
```



```
// B.h
#ifndef _B_H
#define _B_H
#include "A.h"
class B {
public:
    void service(A a);
private:
    A m_aComponent;
};
#endif
```

```
// A.cpp
#include "A.h"
void A::service(B& b) {
    ...
}
```

```
// B.cpp
#include "B.h"
void B::service(A a) {
    ...
}
```

- 編譯 A.cpp 時編譯器說 B.h 裡面 A 是沒有定義過的
- 編譯 B.cpp 時編譯器說 A.h 裡面 B 是沒有定義過的

出現什麼狀況？

```
// A.h
#ifndef _A_H
#define _A_H
#include "B.h"
class A {
public:
    void service(B& b);
private:
    B *m_bptr;
};
#endif
```

```
// B.h
#ifndef _B_H
#define _B_H
#include "A.h"
class B {
public:
    void service(A a);
private:
    A m_aComponent;
};
#endif
```

```
// A.cpp
#include "A.h"
void A::service(B& b) {
    ...
}
```

```
// B.cpp
#include "B.h"
void B::service(A a) {
    ...
}
```

- 編譯 A.cpp 時編譯器說 B.h 裡面 A 是沒有定義過的
 - 編譯 B.cpp 時編譯器說 A.h 裡面 B 是沒有定義過的
- 天啊!!! 不是「用到誰引入誰的定義」嗎？這是什麼狀況？

出現什麼狀況？

```
// A.h
#ifndef _A_H
#define _A_H
#include "B.h"
class A {
public:
    void service(B& b);
private:
    B *m_bptr;
};
#endif
```

```
// B.h
#ifndef _B_H
#define _B_H
#include "A.h"
class B {
public:
    void service(A a);
private:
    A m_aComponent;
};
#endif
```

```
// A.cpp
#include "A.h"
void A::service(B& b) {
    ...
}
```

```
// B.cpp
#include "B.h"
void B::service(A a) {
    ...
}
```

- 編譯 A.cpp 時編譯器說 B.h 裡面 A 是沒有定義過的
 - 編譯 B.cpp 時編譯器說 A.h 裡面 B 是沒有定義過的
- 天啊!!! 不是「用到誰引入誰的定義」嗎？這是什麼狀況？
- 這種 A.h 引入 B.h, 同時 B.h 又引入 A.h 的狀況叫做 **Circular Dependency**, 也許你發現這種編譯錯誤和防止重複引入的 `#ifndef` `#define` `#endif` 有一些關聯性

拿掉 **#ifndef** 相關敘述就好了嗎？

拿掉 `#ifndef` 相關敘述就好了嗎？

- 當然不是

拿掉 `#ifndef` 相關敘述就好了嗎？

- 當然不是
- 拿掉以後更糟，原來能夠解決的「重複引入」問題又出現了

拿掉 `#ifndef` 相關敘述就好了嗎？

- 當然不是
- 拿掉以後更糟，原來能夠解決的「重複引入」問題又出現了
 - 編譯 `A.cpp` 時編譯器說 `fatal error C1014: Include 檔太多：深度 = 1024`

拿掉 `#ifndef` 相關敘述就好了嗎？

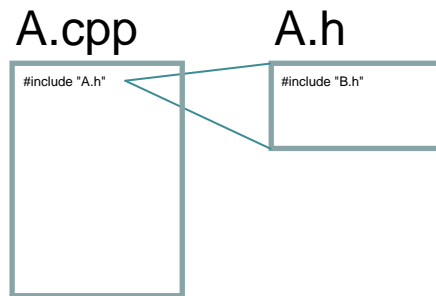
- 當然不是
- 拿掉以後更糟，原來能夠解決的「重複引入」問題又出現了
 - 編譯 `A.cpp` 時編譯器說 `fatal error C1014: Include 檔太多：深度 = 1024`

A.cpp



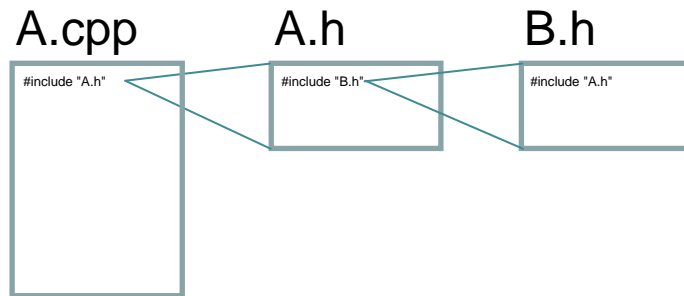
拿掉 `#ifndef` 相關敘述就好了嗎？

- 當然不是
- 拿掉以後更糟，原來能夠解決的「重複引入」問題又出現了
 - 編譯 `A.cpp` 時編譯器說 `fatal error C1014: Include 檔太多：深度 = 1024`



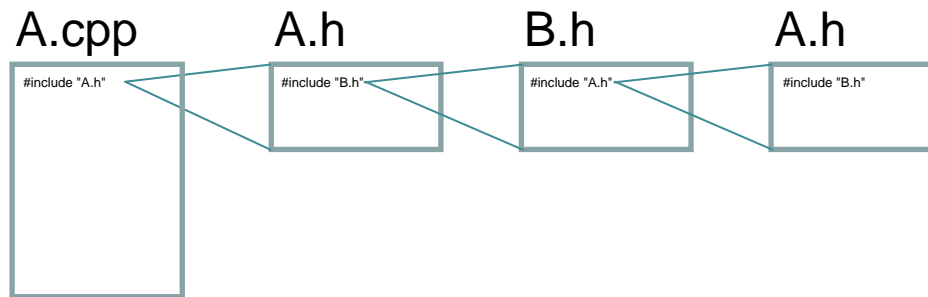
拿掉 `#ifndef` 相關敘述就好了嗎？

- 當然不是
- 拿掉以後更糟，原來能夠解決的「重複引入」問題又出現了
 - 編譯 A.cpp 時編譯器說 fatal error C1014: Include 檔太多：深度 = 1024



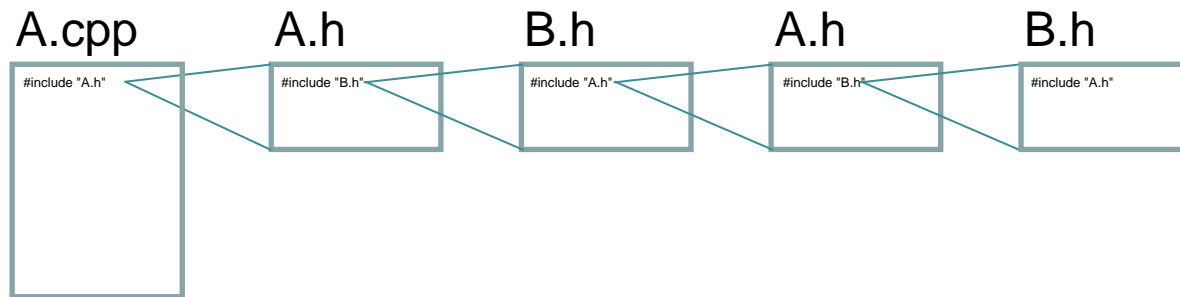
拿掉 `#ifndef` 相關敘述就好了嗎？

- 當然不是
- 拿掉以後更糟，原來能夠解決的「重複引入」問題又出現了
 - 編譯 A.cpp 時編譯器說 fatal error C1014: Include 檔太多：深度 = 1024



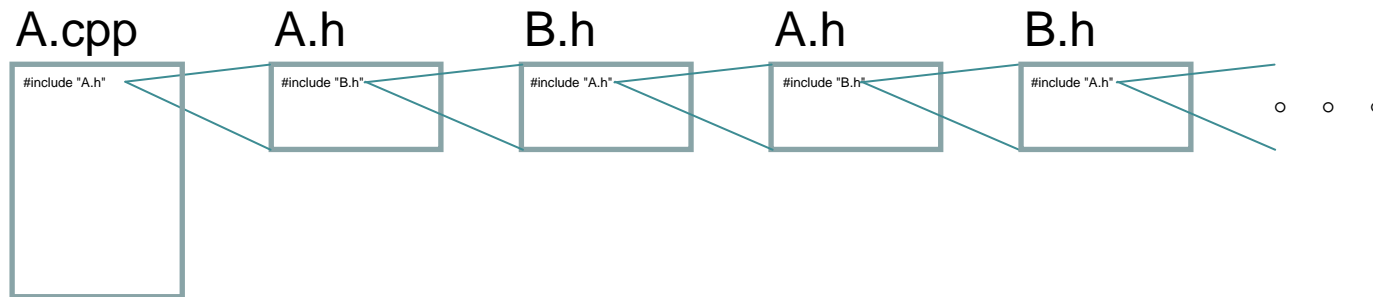
拿掉 `#ifndef` 相關敘述就好了嗎？

- 當然不是
- 拿掉以後更糟，原來能夠解決的「重複引入」問題又出現了
 - 編譯 A.cpp 時編譯器說 fatal error C1014: Include 檔太多：深度 = 1024



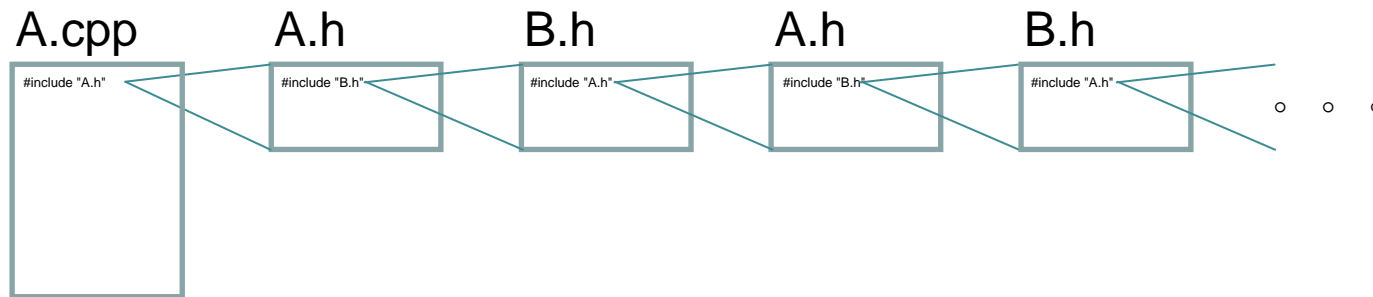
拿掉 `#ifndef` 相關敘述就好了嗎？

- 當然不是
- 拿掉以後更糟，原來能夠解決的「重複引入」問題又出現了
 - 編譯 A.cpp 時編譯器說 fatal error C1014: Include 檔太多：深度 = 1024



拿掉 `#ifndef` 相關敘述就好了嗎？

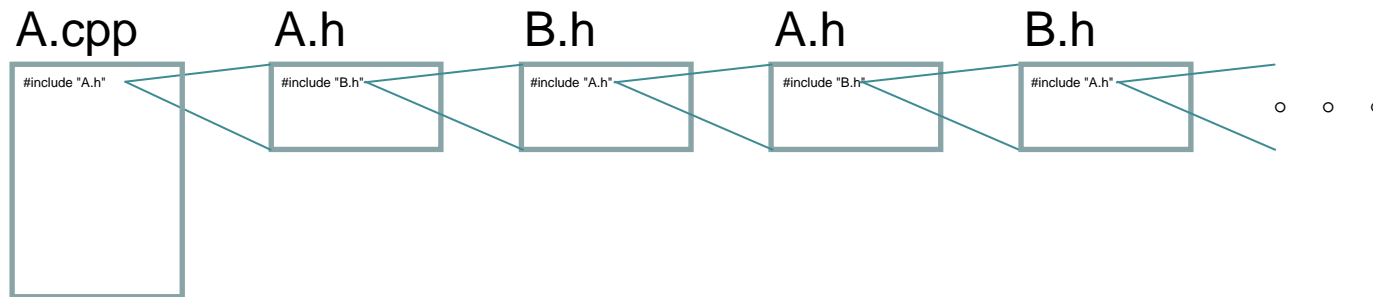
- 當然不是
- 拿掉以後更糟，原來能夠解決的「重複引入」問題又出現了
 - 編譯 `A.cpp` 時編譯器說 fatal error C1014: Include 檔太多：深度 = 1024



- 編譯 `B.cpp` 時編譯器說 fatal error C1014: Include 檔太多：深度 = 1024

拿掉 `#ifndef` 相關敘述就好了嗎？

- 當然不是
- 拿掉以後更糟，原來能夠解決的「重複引入」問題又出現了
 - 編譯 `A.cpp` 時編譯器說 fatal error C1014: Include 檔太多：深度 = 1024



- 編譯 `B.cpp` 時編譯器說 fatal error C1014: Include 檔太多：深度 = 1024
- 還沒有弄清楚狀況前，別急著動手賭自己的運氣

Circular Dependency

Circular Dependency

- 其實這不是多檔案的專案才有的問題, 寫在單一檔案裡的程式也會發生這個問題

Circular Dependency

- 其實這不是多檔案的專案才有的問題，寫在單一檔案裡的程式也會發生這個問題

```
class A {  
public:  
    void service(B& b, B c);  
private:  
    B *m_bptr;  
};  
class B {  
public:  
    void service(A a);  
private:  
    A m_aComponent;  
};  
void A::service(B& b) {  
    ...  
}  
void B::service(A a) {  
    ...  
}
```

Circular Dependency

- 其實這不是多檔案的專案才有的問題，寫在單一檔案裡的程式也會發生這個問題，編譯器看到 **B**& b, **B** c 以及 **B*** m_bptr; 時會說 **B** 沒有定義過

```
class A {  
public:  
    void service(B& b, B c);  
private:  
    B *m_bptr;  
};  
class B {  
public:  
    void service(A a);  
private:  
    A m_aComponent;  
};  
void A::service(B& b) {  
    ...  
}  
void B::service(A a) {  
    ...  
}
```

Circular Dependency

- 其實這不是多檔案的專案才有的問題，寫在單一檔案裡的程式也會發生這個問題，編譯器看到 `B& b`, `B c` 以及 `B* m_bptr`; 時會說 `B` 沒有定義過
- 如果把 `class B` 的定義搬到 `class A` 之前，似乎就可以解決這個問題

```
class A {  
public:  
    void service(B& b, B c);  
private:  
    B *m_bptr;  
};  
class B {  
public:  
    void service(A a);  
private:  
    A m_aComponent;  
};  
void A::service(B& b) {  
    ...  
}  
void B::service(A a) {  
    ...  
}
```

Circular Dependency

- 其實這不是多檔案的專案才有的問題，寫在單一檔案裡的程式也會發生這個問題，編譯器看到 `B& b`, `B c` 以及 `B* m_bptr`; 時會說 `B` 沒有定義過
- 如果把 `class B` 的定義搬到 `class A` 之前，似乎就可以解決這個問題，但是編譯器在看到 `A a` 以及 `A m_aComponent` 的時候又說 `A` 是沒有定義過的

```
class A {  
public:  
    void service(B& b, B c);  
private:  
    B *m_bptr;  
};  
class B {  
public:  
    void service(A a);  
private:  
    A m_aComponent;  
};  
void A::service(B& b) {  
    ...  
}  
void B::service(A a) {  
    ...  
}
```

Circular Dependency

- 其實這不是多檔案的專案才有的問題，寫在單一檔案裡的程式也會發生這個問題，編譯器看到 **B**& b, **B** c 以及 **B*** m_bptr; 時會說 **B** 沒有定義過
- 如果把 class B 的定義搬到 class A 之前，似乎就可以解決這個問題，但是編譯器在看到 **A** a 以及 **A** m_aComponent 的時候又說 **A** 是沒有定義過的
- 這樣子有點像是「雞生蛋，蛋生雞」的問題，好像完全無解了？

```
class A {  
public:  
    void service(B& b, B c);  
private:  
    B *m_bptr;  
};  
class B {  
public:  
    void service(A a);  
private:  
    A m_aComponent;  
};  
void A::service(B& b) {  
    ...  
}  
void B::service(A a) {  
    ...  
}
```

Circular Dependency

- 其實這不是多檔案的專案才有的問題，寫在單一檔案裡的程式也會發生這個問題，編譯器看到 `B& b`, `B c` 以及 `B* m_bptr`; 時會說 `B` 沒有定義過
- 如果把 `class B` 的定義搬到 `class A` 之前，似乎就可以解決這個問題，但是編譯器在看到 `A a` 以及 `A m_aComponent` 的時候又說 `A` 是沒有定義過的
- 這樣子有點像是「雞生蛋，蛋生雞」的問題，好像完全無解了？
- 解決的方法是在 `class A` 定義之前加上 `class B;` 的預先宣告敘述 (forward declaration)

```
class A {  
public:  
    void service(B& b, B c);  
private:  
    B *m_bptr;  
};  
class B {  
public:  
    void service(A a);  
private:  
    A m_aComponent;  
};  
void A::service(B& b) {  
    ...  
}  
void B::service(A a) {  
    ...  
}
```


Circular Dependency

- 其實這不是多檔案的專案才有的問題，寫在單一檔案裡的程式也會發生這個問題，編譯器看到 `B& b`, `B c` 以及 `B* m_bptr`; 時會說 `B` 沒有定義過
- 如果把 `class B` 的定義搬到 `class A` 之前，似乎就可以解決這個問題，但是編譯器在看到 `A a` 以及 `A m_aComponent` 的時候又說 `A` 是沒有定義過的
- 這樣子有點像是「雞生蛋，蛋生雞」的問題，好像完全無解了？
- 解決的方法是在 `class A` 定義之前加上 `class B;` 的預先宣告敘述 (forward declaration)

```
class B;  
class A {  
public:  
    void service(B& b, B c);  
private:  
    B *m_bptr;  
};  
class B {  
public:  
    void service(A a);  
private:  
    A m_aComponent;  
};  
void A::service(B& b) {  
    ...  
}  
void B::service(A a) {  
    ...  
}
```

Circular Dependency

- 其實這不是多檔案的專案才有的問題，寫在單一檔案裡的程式也會發生這個問題，編譯器看到 **B**& b, **B** c 以及 **B*** m_bptr; 時會說 **B** 沒有定義過
- 如果把 **class B** 的定義搬到 **class A** 之前，似乎就可以解決這個問題，但是編譯器在看到 **A** a 以及 **A** m_aComponent 的時候又說 **A** 是沒有定義過的
- 這樣子有點像是「雞生蛋，蛋生雞」的問題，好像完全無解了？
- 解決的方法是在 **class A** 定義之前加上 **class B;** 的預先宣告敘述 (forward declaration), 如此編譯器在看到 **B**& b, **B** c 以及 **B***m_bptr 時知道 **B** 是一個類別

```
class B;  
class A {  
public:  
    void service(B& b, B c);  
private:  
    B *m_bptr;  
};  
class B {  
public:  
    void service(A a);  
private:  
    A m_aComponent;  
};  
void A::service(B& b) {  
    ...  
}  
void B::service(A a) {  
    ...  
}
```

Circular Dependency

- 其實這不是多檔案的專案才有的問題，寫在單一檔案裡的程式也會發生這個問題，編譯器看到 `B& b`, `B c` 以及 `B* m_bptr` 時會說 `B` 沒有定義過
- 如果把 `class B` 的定義搬到 `class A` 之前，似乎就可以解決這個問題，但是編譯器在看到 `A a` 以及 `A m_aComponent` 的時候又說 `A` 是沒有定義過的
- 這樣子有點像是「雞生蛋，蛋生雞」的問題，好像完全無解了？
- 解決的方法是在 `class A` 定義之前加上 `class B;` 的預先宣告敘述 (forward declaration), 如此編譯器在看到 `B& b`, `B c` 以及 `B* m_bptr` 時知道 `B` 是一個類別，因為在類別 `A` 定義時只有用到 `B` 類別的參考/指標/參數，編譯器不需要知道 `B` 類別的細節

```
class B;
class A {
public:
    void service(B& b, B c);
private:
    B *m_bptr;
};
class B {
public:
    void service(A a);
private:
    A m_aComponent;
};
void A::service(B& b) {
    ...
}
void B::service(A a) {
    ...
}
```

Circular Dependency (cont'd)

Circular Dependency (cont'd)

- 預先宣告 (forward declaration) 敘述不是隨便都可以用的, 定義指標或是參考變數時可以使用

```
class B;  
B *ptr = &bObj;  
B& bRef = bObj;
```

Circular Dependency (cont'd)

- 預先宣告 (forward declaration) 敘述不是隨便都可以用的, 定義指標或是參考變數時可以使用, 定義物件時不能使用預先宣告敘述

```
class B;  
B *ptr = &bObj;  
B& bRef = bObj;  
B bObj; // error
```

Circular Dependency (cont'd)

- 預先宣告 (forward declaration) 敘述不是隨便都可以用的, 定義指標或是參考變數時可以使用, 定義物件時不能使用預先宣告敘述

```
class B;  
B *ptr = &bObj;  
B& bRef = bObj;  
B bObj; // error
```

```
class B {  
public:  
    void service(A a);  
private:  
    A m_aComponent;  
};  
B bObj;
```

- 編譯器需要曉得完整的類別定義, 才能夠定義這個類別的物件, 例如上圖的 bObj

Circular Dependency (cont'd)

- 預先宣告 (forward declaration) 敘述不是隨便都可以用的, 定義指標或是參考變數時可以使用, 定義物件時不能使用預先宣告敘述

```
class B;  
B *ptr = &bObj;  
B& bRef = bObj;  
B bObj; // error
```

```
class B {  
public:  
    void service(A a);  
private:  
    A m_aComponent;  
};  
B bObj;
```

- 編譯器需要曉得完整的類別定義, 才能夠定義這個類別的物件, 例如上圖的 bObj
- 類別 A 和 B 有循環關聯性的時後, 有的時候好像會使得類別 A 和 B 完全沒有辦法定義, 如右圖程式:

```
class A {  
public:  
    void service(B b);  
private:  
    B *m_bptr;  
};  
class B {  
public:  
    void service(A a);  
private:  
    A m_aComponent;  
};  
void A::service(B b) {  
    ...  
}  
void B::service(A a) {  
    ...  
}
```


Circular Dependency (cont'd)

- 預先宣告 (forward declaration) 敘述不是隨便都可以用的, 定義指標或是參考變數時可以使用, 定義物件時不能使用預先宣告敘述

```
class B;  
B *ptr = &bObj;  
B& bRef = bObj;  
B bObj; // error
```

```
class B {  
public:  
    void service(A a);  
private:  
    A m_aComponent;  
};  
B bObj;
```

- 編譯器需要曉得完整的類別定義, 才能夠定義這個類別的物件, 例如上圖的 bObj
- 類別 A 和 B 有循環關聯性的時後, 有的時候好像會使得類別 A 和 B 完全沒有辦法定義, 如右圖程式: 其實也是加上預先宣告以後就沒有問題了

```
class B;  
class A {  
public:  
    void service(B b);  
private:  
    B *m_bptr;  
};  
class B {  
public:  
    void service(A a);  
private:  
    A m_aComponent;  
};  
void A::service(B b) {  
    ...  
}  
void B::service(A a) {  
    ...  
}
```

Circular Dependency (cont'd)

- 預先宣告 (forward declaration) 敘述不是隨便都可以用的, 定義指標或是參考變數時可以使用, 定義物件時不能使用預先宣告敘述

```
class B;  
B *ptr = &bObj;  
B& bRef = bObj;  
B bObj; // error
```

```
class B {  
public:  
    void service(A a);  
private:  
    A m_aComponent;  
};  
B bObj;
```

- 編譯器需要曉得完整的類別定義, 才能夠定義這個類別的物件, 例如上圖的 bObj
- 類別 A 和 B 有循環關聯性的時候, 有的時候好像會使得類別 A 和 B 完全沒有辦法定義, 如右圖程式: 其實也是加上預先宣告以後就沒有問題了, A::service(B b) 裡面的 b 雖然不是指標, 但只是函式原型

```
class B;  
class A {  
public:  
    void service(B b);  
private:  
    B *m_bptr;  
};  
class B {  
public:  
    void service(A a);  
private:  
    A m_aComponent;  
};  
void A::service(B b) {  
    ...  
}  
void B::service(A a) {  
    ...  
}
```

Circular Dependency (cont'd)

```
class A {  
public:  
    void service(B* bPtr);  
private:  
    B m_bComponent;  
};  
class B {  
public:  
    void service(A a);  
private:  
    A m_aComponent;  
};  
void A::service(B* bPtr) {  
    ...  
}  
void B::service(A a) {  
    ...  
}
```

Circular Dependency (cont'd)

- 右側這個程式一定無法編譯, 就算使用預先宣告都沒有辦法編譯, 但是別擔心

```
class A {  
public:  
    void service(B* bPtr);  
private:  
    B m_bComponent;  
};  
class B {  
public:  
    void service(A a);  
private:  
    A m_aComponent;  
};  
void A::service(B* bPtr) {  
    ...  
}  
void B::service(A a) {  
    ...  
}
```

Circular Dependency (cont'd)

- 右側這個程式一定無法編譯, 就算使用預先宣告都沒有辦法編譯, 但是別擔心, 這是不可能發生的, 因為 A 類別裡面有 B 類別的物件成員, 同時 B 類別裡面有 A 類別的物件成員, 會導致需要無窮大記憶體空間的物件

```
class A {  
public:  
    void service(B* bPtr);  
private:  
    B m_bComponent;  
};  
class B {  
public:  
    void service(A a);  
private:  
    A m_aComponent;  
};  
void A::service(B* bPtr) {  
    ...  
}  
void B::service(A a) {  
    ...  
}
```

Circular Dependency (cont'd)

- 右側這個程式一定無法編譯, 就算使用預先宣告都沒有辦法編譯, 但是別擔心, 這是不可能發生的, 因為 A 類別裡面有 B 類別的物件成員, 同時 B 類別裡面有 A 類別的物件成員, 會導致需要無窮大記憶體空間的物件 (就像 A 類別裡面有 A 類別的物件成員一樣)

```
class A {  
public:  
    void service(B* bPtr);  
private:  
    B m_bComponent;  
};  
class B {  
public:  
    void service(A a);  
private:  
    A m_aComponent;  
};  
void A::service(B* bPtr) {  
    ...  
}  
void B::service(A a) {  
    ...  
}
```

Circular Dependency (cont'd)

- 右側這個程式一定無法編譯, 就算使用預先宣告都沒有辦法編譯, 但是別擔心, 這是不可能發生的, 因為 A 類別裡面有 B 類別的物件成員, 同時 B 類別裡面有 A 類別的物件成員, 會導致需要無窮大記憶體空間的物件 (就像 A 類別裡面有 A 類別的物件成員一樣)
- 一般情形下如果 B 類別裡面有 A 類別的成員, A 類別裡就只能有 B 類別的指標或是參考, 如此就可以使用預先宣告敘述來解決這個問題了

```
class A {  
public:  
    void service(B* bPtr);  
private:  
    B m_bComponent;  
};  
class B {  
public:  
    void service(A a);  
private:  
    A m_aComponent;  
};  
void A::service(B* bPtr) {  
    ...  
}  
void B::service(A a) {  
    ...  
}
```

回到多檔案的循環依賴

```
// A.h
#ifndef _A_H
#define _A_H
#include "B.h"
class A {
public:
    void service(B& b);
private:
    B *m_bptr;
};
#endif
```

```
// A.cpp
#include "A.h"
void A::service(B& b) {
    ...
}
```

```
// B.h
#ifndef _B_H
#define _B_H
#include "A.h"
class B {
public:
    void service(A a);
private:
    A m_aComponent;
};
#endif
```

```
// B.cpp
#include "B.h"
void B::service(A a) {
    ...
}
```


回到多檔案的循環依賴

```
// A.h
#ifndef _A_H
#define _A_H
#include "B.h"
class A {
public:
    void service(B& b);
private:
    B *m_bptr;
};
#endif
```

```
// B.h
#ifndef _B_H
#define _B_H
#include "A.h"
class B {
public:
    void service(A a);
private:
    A m_aComponent;
};
#endif
```

```
// A.cpp
#include "A.h"
void A::service(B& b) {
    ...
}
```

```
// B.cpp
#include "B.h"
void B::service(A a) {
    ...
}
```

- 編譯 A.cpp 時編譯器的錯誤訊息是
B.h 裡面 A 是沒有定義過的

回到多檔案的循環依賴

```
// A.h
#ifndef _A_H
#define _A_H
#include "B.h"
class A {
public:
    void service(B& b);
private:
    B *m_bptr;
};
#endif
```

```
// B.h
#ifndef _B_H
#define _B_H
#include "A.h"
class B {
public:
    void service(A a);
private:
    A m_aComponent;
};
#endif
```

```
// A.cpp
#include "A.h"
void A::service(B& b) {
    ...
}
```

```
// B.cpp
#include "B.h"
void B::service(A a) {
    ...
}
```

- 編譯 A.cpp 時編譯器的錯誤訊息是
B.h 裡面 A 是沒有定義過的

```
// A.cpp
```

```
#include "A.h"
```

```
void A::service(B& b) {
    ...
}
```

回到多檔案的循環依賴

```
// A.h
#ifndef _A_H
#define _A_H
#include "B.h"
class A {
public:
    void service(B& b);
private:
    B *m_bptr;
};
#endif
```

```
// B.h
#ifndef _B_H
#define _B_H
#include "A.h"
class B {
public:
    void service(A a);
private:
    A m_aComponent;
};
#endif
```

```
// A.cpp
#include "A.h"
void A::service(B& b) {
    ...
}
```

```
// B.cpp
#include "B.h"
void B::service(A a) {
    ...
}
```

- 編譯 A.cpp 時編譯器的錯誤訊息是
B.h 裡面 A 是沒有定義過的

```
// A.cpp
// A.h
#ifndef _A_H
#define _A_H

#include "B.h"

class A {
public:
    void service(B& b);
private:
    B *m_bptr;
};
#endif
void A::service(B& b) {
    ...
}
```

回到多檔案的循環依賴

```
// A.h
#ifndef _A_H
#define _A_H
#include "B.h"
class A {
public:
    void service(B& b);
private:
    B *m_bptr;
};
#endif
```

```
// A.cpp
#include "A.h"
void A::service(B& b) {
    ...
}
```

```
// B.h
#ifndef _B_H
#define _B_H
#include "A.h"
class B {
public:
    void service(A a);
private:
    A m_aComponent;
};
#endif
```

```
// B.cpp
#include "B.h"
void B::service(A a) {
    ...
}
```

- 編譯 A.cpp 時編譯器的錯誤訊息是
B.h 裡面 A 是沒有定義過的

```
// A.cpp
// A.h
#ifndef _A_H
#define _A_H

// B.h
#ifndef _B_H
#define _B_H

#include "A.h"

class B {
public:
    void service( A a );
private:
    A m_aComponent;
};
#endif

class A {
public:
    void service(B& b);
private:
    B *m_bptr;
};
#endif
void A::service(B& b) {
    ...
}
```

回到多檔案的循環依賴

```
// A.h
#ifndef _A_H
#define _A_H
#include "B.h"
class A {
public:
    void service(B& b);
private:
    B *m_bptr;
};
#endif
```

```
// A.cpp
#include "A.h"
void A::service(B& b) {
    ...
}
```

```
// B.h
#ifndef _B_H
#define _B_H
#include "A.h"
class B {
public:
    void service(A a);
private:
    A m_aComponent;
};
#endif
```

```
// B.cpp
#include "B.h"
void B::service(A a) {
    ...
}
```

- 編譯 A.cpp 時編譯器的錯誤訊息是
B.h 裡面 A 是沒有定義過的

```
// A.cpp
// A.h
#ifndef _A_H
#define _A_H

// B.h
#ifndef _B_H
#define _B_H

// A.h
#ifndef _A_H
...
#endif
class B {
public:
    void service( A a );
private:
    A m_aComponent;
};
#endif

class A {
public:
    void service(B& b);
private:
    B *m_bptr;
};
#endif

void A::service(B& b) {
    ...
}
```

回到多檔案的循環依賴

```
// A.h
#ifndef _A_H
#define _A_H
#include "B.h"
class A {
public:
    void service(B& b);
private:
    B *m_bptr;
};
#endif
```

```
// A.cpp
#include "A.h"
void A::service(B& b) {
    ...
}
```

```
// B.h
#ifndef _B_H
#define _B_H
#include "A.h"
class B {
public:
    void service(A a);
private:
    A m_aComponent;
};
#endif
```

```
// B.cpp
#include "B.h"
void B::service(A a) {
    ...
}
```

- 編譯 A.cpp 時編譯器的錯誤訊息是
B.h 裡面 A 是沒有定義過的

```
// A.cpp
// A.h
#ifndef _A_H
#define _A_H

// B.h
#ifndef _B_H
#define _B_H

// A.h
#ifndef _A_H
...
#endif
class B {
public:
    void service(A a);
private:
    A m_aComponent;
};
#endif

class A {
public:
    void service(B& b);
private:
    B *m_bptr;
};
#endif

void A::service(B& b) {
    ...
}
```

回到多檔案的循環依賴

```
// A.h
#ifndef _A_H
#define _A_H
#include "B.h"
class A {
public:
    void service(B& b);
private:
    B *m_bptr;
};
#endif
```

```
// A.cpp
#include "A.h"
void A::service(B& b) {
    ...
}
```

```
// B.h
#ifndef _B_H
#define _B_H
#include "A.h"
class B {
public:
    void service(A a);
private:
    A m_aComponent;
};
#endif
```

```
// B.cpp
#include "B.h"
void B::service(A a) {
    ...
}
```

- 編譯 A.cpp 時編譯器的錯誤訊息是
B.h 裡面 A 是沒有定義過的
- 預先宣告是唯一的解決方法

```
// A.cpp
// A.h
#ifndef _A_H
#define _A_H

// B.h
#ifndef _B_H
#define _B_H

// A.h
#ifndef _A_H
...
#endif

class B {
public:
    void service(A a);
private:
    A m_aComponent;
};
#endif

class A {
public:
    void service(B& b);
private:
    B *m_bptr;
};
#endif

void A::service(B& b) {
    ...
}
```

回到多檔案的循環依賴

```
// A.h
#ifndef _A_H
#define _A_H
class B;
class A {
public:
    void service(B& b);
private:
    B *m_bptr;
};
#endif
```

```
// A.cpp
#include "A.h"
void A::service(B& b) {
    ...
}
```

```
// B.h
#ifndef _B_H
#define _B_H
#include "A.h"
class B {
public:
    void service(A a);
private:
    A m_aComponent;
};
#endif
```

```
// B.cpp
#include "B.h"
void B::service(A a) {
    ...
}
```

- 編譯 A.cpp 時編譯器的錯誤訊息是
B.h 裡面 A 是沒有定義過的
- 預先宣告是唯一的解決方法

```
// A.cpp
// A.h
#ifndef _A_H
#define _A_H

// B.h
#ifndef _B_H
#define _B_H
// A.h
#ifndef _A_H
...
#endif
class B {
public:
    void service(A a);
private:
    A m_aComponent;
};
#endif
class A {
public:
    void service(B& b);
private:
    B *m_bptr;
};
#endif
void A::service(B& b) {
    ...
}
```