

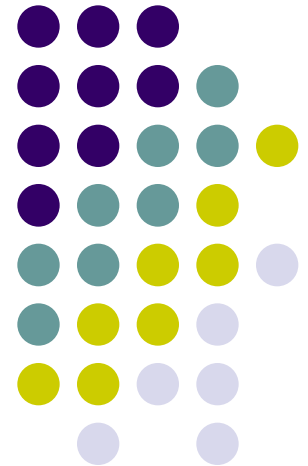
第十四章

解構元與拷貝建構元

認識解構元

學習動態記憶體配置與解構元的關係

使用拷貝建構元





解構元

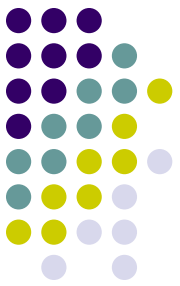
- 建構元是在物件初次被建立時呼叫
- 解構元是在物件被銷毀（ destroy ）時呼叫
- 銷毀指的是釋放物件原先所佔有的記憶空間
- 解構元的名稱和類別的名稱相同，之前必須加上一個 ~（ tilde ）符號
- 解構元的定義格式

```
~類別名稱()  
{  
    程式敘述 ;  
    ...  
}
```

解構元的名稱必須和類別名稱相同

解構元不能傳入任何引數

解構元沒有傳回值



解構元的使用 (1/2)

- 下面的範例裡加入一個解構元，以便觀察它的運作

```
01 // prog14_1, 解構元的使用
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 class CWin // 定義視窗類別 CWin
06 {
07     private:
08         char id;
09         int width, height;
10
11     public:
12         CWin(char i,int w,int h):id(i),width(w),height(h)
13         {
14             cout << "建構元被呼叫了..." << endl;
15         }
16         ~CWin() // 解構元
17         {
18             cout << "解構元被呼叫了， win " << this->id << "被銷毀了.." << endl;
19             system("pause");
20         }
```



解構元的使用 (2/2)

```

21     void show member(void)
22     {
23         cout << "Window " << id << ": ";
24         cout << "width=" << width << ", height=" << height << endl;
25     }
26 };

```

```

28 int main(void)

```

```

29 {
30     CWin win1('A', 50, 40);
31     CWin win2('B', 40, 50);
32     CWin win3('C', 60, 70);
33     CWin win4('D', 90, 40);
34
35     win1.show member();
36     win2.show member();
37
38     system("pause");
39     return 0;
40 }

```

/* prog14_1 OUTPUT-----

建構元被呼叫了...

建構元被呼叫了...

建構元被呼叫了...

建構元被呼叫了...

Window A: width=50, height=40

Window B: width=40, height=50

請按任意鍵繼續 . . . ----- 執行第 38 行的結果

解構元被呼叫了, Win D 被銷毀了.. ----- win4 被銷毀, 這是執行第 18 行的結果

請按任意鍵繼續 . . . ----- 執行第 19 行的結果

解構元被呼叫了, Win C 被銷毀了.. ----- win3 被銷毀, 這是執行第 18 行的結果

請按任意鍵繼續 . . . ----- 執行第 19 行的結果

解構元被呼叫了, Win B 被銷毀了.. ----- win2 被銷毀, 這是執行第 18 行的結果

請按任意鍵繼續 . . . ----- 執行第 19 行的結果

解構元被呼叫了, Win A 被銷毀了.. ----- win1 被銷毀, 這是執行第 18 行的結果

請按任意鍵繼續 . . . ----- 執行第 19 行的結果

-----*/



解構元的位置

- 在類別內部宣告解構元的原型

```
~CWin();    // 解構元的原型
```

- 在類別外面定義解構元時，要指明其所屬的建構元

```
CWin::~~CWin()  
{  
    // 解構元的程式碼  
}
```



固定空間的記憶體配置 (1/2)

- 先看一個簡單的例子，此例無關動態記憶體配置

```
01 // prog14_2, 固定空間的記憶體配置
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 class CWin // 定義視窗類別 CWin
06 {
07     private:
08         char id,title[20];
09
10     public:
11         CWin(char i='D', char *text="Default window"):id(i)
12         {
13             strcpy(title,text); // 將 text 指向的字串拷貝到 title 陣列裡
14         }
15         ~CWin() // 解構元
16         {
17             cout << "解構元被呼叫了，win " << this->id << "被銷毀了.." << endl;
18             system("pause");
19         }
```



固定空間的記憶體配置 (2/2)

```

20     void show(void)           // 顯示id與title 成員
21     {
22         cout << "Window " << id << ": " << title << endl;
23     }

```

```

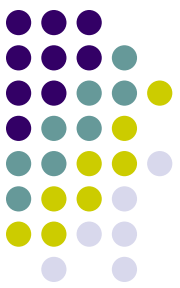
24 };
25
26 int main(void)
27 {
28     CWin win1('A',"Main window");
29     CWin win2('B');
30
31     win1.show();
32     win2.show();
33
34     cout << "sizeof(win1)= " << sizeof(win1) << endl;
35     cout << "sizeof(win2)= " << sizeof(win2) << endl;
36
37     system("pause");
38     return 0;
39 }

```

/* prog14_2 OUTPUT-----

Window A: Main window
Window B: Default window
sizeof(win1)= 21
sizeof(win2)= 21
請按任意鍵繼續 . . .
解構元被呼叫了，Win B 被銷毀了..
請按任意鍵繼續 . . .
解構元被呼叫了，Win A 被銷毀了..
請按任意鍵繼續 . . .

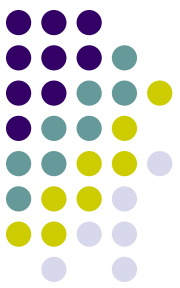
-----*/



使用動態記憶體配置 (1/3)

- 下面的範例將prog14_2改以動態的方式來配置記憶體

```
01 // prog14_3, 使用動態記憶體配置
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 class CWin // 定義視窗類別 CWin
06 {
07     private:
08         char id, *title; // 宣告 title 為指向字元陣列的指標
09
10     public:
11         CWin(char i='D', char *text="Default window"):id(i)
12         {
13             title=new char[strlen(text)+1]; // 配置記憶體空間
14             strcpy(title,text);
15         }
16         ~CWin() // 解構元的原型
17         {
18             cout << "解構元被呼叫了，Win " << this->id << "被銷毀了.." << endl;
19             delete [] title; // 釋放 title 所指向的記憶體空間
20             system("pause");
21         }
```

使用動態記憶體配置 (2/3)

● 原始程式編譯及連結的過程

```

22     void show(void)
23     {
24         cout << "Window " << id << ": " << title << endl;
25     }
26 };
27
28 int main(void)
29 {
30     CWin win1('A', "Main window");
31     CWin win2('B');
32
33     win1.show();
34     win2.show();
35     cout << "sizeof(win1)= " << sizeof(win1) << endl;
36     cout << "sizeof(win2)= " << sizeof(win2) << endl;
37
38     system("pause");
39     return 0;
40 }

```

/* prog14_3 OUTPUT-----

```

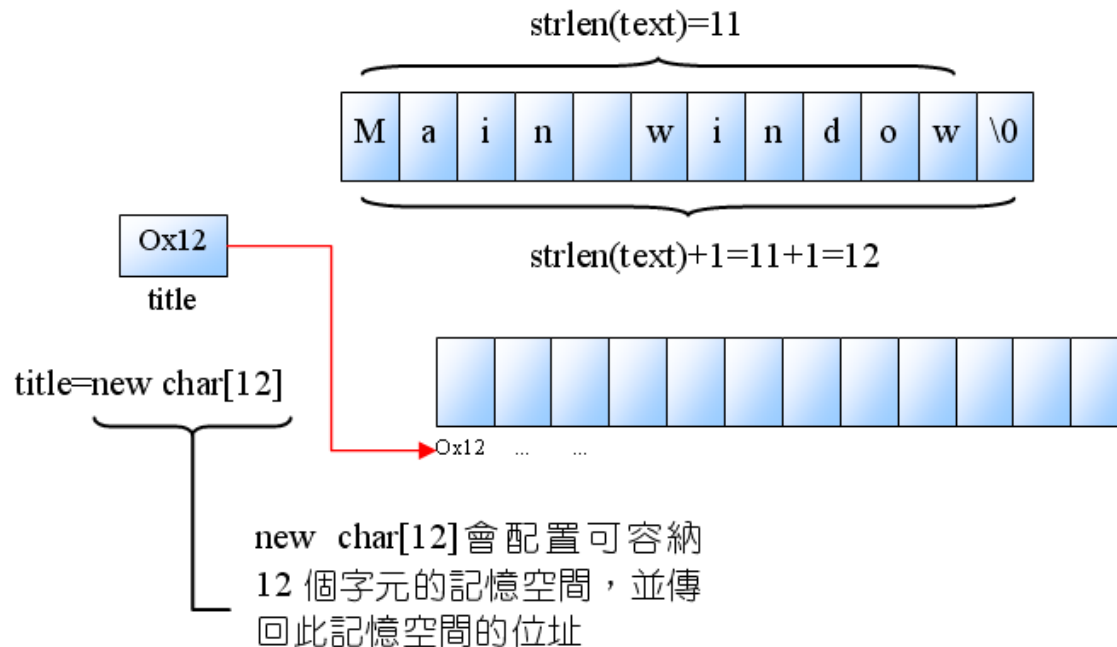
Window A: Main window
Window B: Default window
sizeof(win1)= 8
sizeof(win2)= 8
請按任意鍵繼續 . . .
解構元被呼叫了, Win B 被銷毀了..
請按任意鍵繼續 . . .
解構元被呼叫了, Win A 被銷毀了..
請按任意鍵繼續 . . .
-----*/

```



使用動態記憶體配置 (3/3)

- 下圖為prog14_3中，記憶空間配置過程的說明





錯誤的使用動態記憶體配置 (1/2)

- 下面的範例修改自prog14_3，這是個錯誤的示範

```
01 // prog14_4, 使用動態記憶體配置, 錯誤的示範
```

```
02 #include <iostream>
```

```
03 #include <cstdlib>
```

```
04 using namespace std;
```

```
05
```

```
06 // 將 prog14_3 CWin 類別的定義放在這裡
```

```
07
```

```
08 int main(void)
```

```
09 {
```

```
10     CWin win1('A', "Main window");
```

```
11     CWin *ptr;
```

```
12     ptr=new CWin('B');
```

```
13
```

```
14     win1.show();
```

```
15     ptr->show();
```

```
16
```

```
17     system("pause");
```

```
18     return 0;
```

```
19 }
```

```
/* prog14_4 OUTPUT-----
```

```
Window A: Main window
```

```
Window B: Defaule window
```

```
請按任意鍵繼續 . . .
```

```
解構元被呼叫了, Win A 被銷毀了..
```

```
請按任意鍵繼續 . . .
```

```
-----*/
```

```
// 宣告 ptr 為指向 CWin 物件的指標
```

```
// 建立新的物件, 並讓 ptr 指向它
```

```
// 以 win1 物件呼叫 show() 函數
```

```
// 以 ptr 指標呼叫 show() 函數
```



錯誤的使用動態記憶體配置 (2/2)

- 下面的範例是更正過後的程式碼

```

01 // prog14_5, 使用動態記憶體配置
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05
06 // 將 prog14_3 CWin 類別的定義放在這裡

```

```

07
08 int main(void)
09 {
10     CWin win1('A', "Main window");
11     CWin *ptr;
12     ptr=new CWin('B');
13
14     win1.show();
15     ptr->show();
16
17     system("pause");
18

```

```

19     delete ptr; // 釋放 ptr 所指向物件之記憶體

```

```

20
21     return 0;
22 }

```

```

/* prog14_5 OUTPUT-----
Window A: Main window
Window B: Defaule window
請按任意鍵繼續 . . .
解構元被呼叫了，Win B 被銷毀了..
請按任意鍵繼續 . . .
解構元被呼叫了，Win A 被銷毀了..
請按任意鍵繼續 . . .
-----*/

```

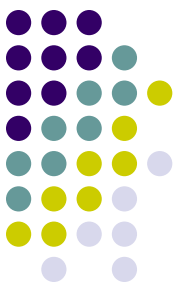


預設的拷貝建構元 (1/3)

- 藉由拷貝建構元，我們可以利用已建立的物件為初值來建立另一個物件

```
CWin win1('A', 50, 40);    // 建立 win1 物件
```

```
CWin win2(win1);          // 以 win1 的內容為初值來建立 win2 物件
```



預設的拷貝建構元 (2/3)

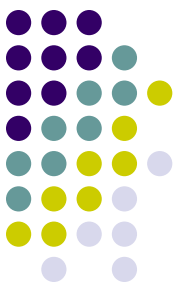
- 下面的範例說明如何在程式中呼叫預設的拷貝建構元

```

01 // prog14_6, 預設的拷貝建構元
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 class CWin // 定義視窗類別 CWin
06 {
07     private:
08         char id;
09         int width, height;
10
11     public:
12         CWin(char i,int w,int h):id(i),width(w),height(h)
13         {
14             cout << "建構元被呼叫了..." <<endl;
15         }
16         void show member(void)
17         {
18             cout << "Window " << id << ": ";
19             cout << "width=" << width << ", height=" << height << endl;
20         }
21 };

```

/* prog14_6 OUTPUT-----
 建構元被呼叫了...
 Window A: width=50, height=40
 Window A: width=50, height=40
-----*/



預設的拷貝建構元 (3/3)

```
22
23 int main(void)
24 {
25     CWin win1('A',50,40);           // 建立 win1 物件
26     CWin win2(win1);               // 呼叫預設的拷貝建構元
27
28     win1.show member();
29     win2.show member();
30
31     system("pause");
32     return 0;
33 }
```

```
/* prog14_6 OUTPUT-----
建構元被呼叫了...
Window A: width=50, height=40
Window A: width=50, height=40
-----*/
```

- 從prog14_6輸出結果可以得知
 - 編譯器幫我們提供拷貝建構元
 - 拷貝建構元可用來拷貝一個已存在物件之成員給新建的物件

26 行呼叫預設拷貝建構元的敘述可改寫成：

```
CWin win2=win1;    // 以 win1 物件的內容為初值來建立 win2 物件
```



撰寫自己的拷貝建構元 (1/3)

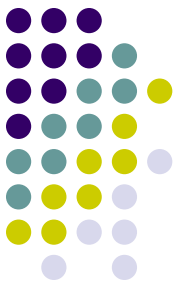
- 要自行提供拷貝建構元，必須以下面的語法來定義

```
類別名稱 (const 類別名稱 & )  
{  
    程式敘述 ;  
    .....  
}
```

拷貝建構元的名稱必須和類別名稱相同

引數必須是物件的「參照」

拷貝建構元沒有傳回值



撰寫自己的拷貝建構元 (2/3)

- 下面的程式碼是加入拷貝建構元的範例

```
01 // prog14_7, 撰寫自己的拷貝建構元
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 class CWin // 定義視窗類別 CWin
06 {
07     private:
08         char id;
09         int width, height;
10
11     public:
12         CWin(char i,int w,int h):id(i),width(w),height(h)
13         {
14             cout << "建構元被呼叫了..." << endl;
15         }
16         CWin(const CWin &win) // 定義拷貝建構元
17         {
18             cout << "拷貝建構元被呼叫了..." << endl;
19             id=win.id;
20             width=win.width;
21             height=win.height;
22         }
```

```
/* prog14_7 OUTPUT-----
建構元被呼叫了...
拷貝建構元被呼叫了...
Window A: width=50, height=40
Window A: width=50, height=40
-----*/
```



撰寫自己的拷貝建構元 (3/3)

```
23     void show member(void)
24     {
25         cout << "Window " << id << ": ";
26         cout << "width=" << width << ", height=" << height << endl;
27     }
28 };
29
30 int main(void)
31 {
32     CWin win1('A',50,40);
33     CWin win2(win1);           // 呼叫拷貝建構元
34
35     win1.show member();
36     win2.show member();
37
38     system("pause");
39     return 0;
40 }
```

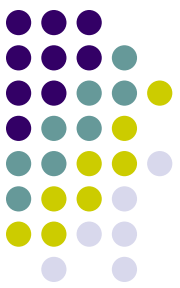
```
/* prog14_7 OUTPUT-----
建構元被呼叫了...
拷貝建構元被呼叫了...
Window A: width=50, height=40
Window A: width=50, height=40
-----*/
```



錯誤的使用拷貝建構元 (1/6)

- 下面的範例修改自prog14_3，其中加入拷貝建構元

```
01 // prog14_8, 錯誤示範，未撰寫拷貝建構元而使用預設的版本
02 #include <iostream >
03 #include <cstdlib>
04 using namespace std;
05 class CWin // 定義視窗類別 CWin
06 {
07     private:
08         char id, *title;
09
10     public:
11         CWin(char i='D', char *text="Default window"):id(i)
12         {
13             cout << "建構元被呼叫了..." << endl;
14             title=new char[strlen(text)+1]; // 配置記憶空間
15             strcpy(title,text);
16         }
17         CWin(const CWin &win)
18         {
19             cout<< "拷貝建構元被呼叫了..." <<endl;
20             id=win.id;
21             title=win.title; } 拷貝資料成員
22 }
```



錯誤的使用拷貝建構元 (2/6)

```

23         ~CWin()                                // 解構元的原型
24     {
25         delete [] title;
26     }
27     void show(void)
28     {
29         cout << "Window " << id << ": " << title << endl;
30     }
31 };
32
33 int main(void)
34 {
35     CWin *ptr1=new CWin('A',"Main window");
36     CWin *ptr2=new CWin(*ptr1);  // 以 ptr1 所指向的物件為初值建立新物件
37
38     ptr1->show();
39     ptr2->show();
40
41     delete ptr1;                                // 釋放 ptr1 所指向的記憶空間
42     cout << "將 ptr1 所指向的物件刪除後..." << endl;
43     ptr2->show();
44
45     delete ptr2;                                // 釋放 ptr2 所指向的記憶空間
46     system("pause");
47     return 0;
48 }

```

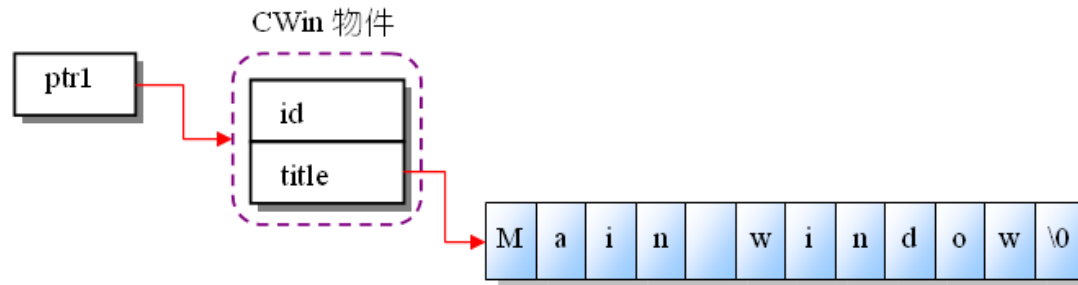
/* prog14_8 OUTPUT-----
 建構元被呼叫了...
 拷貝建構元被呼叫了...
 Window A: Main window
 Window A: Main window
 將 ptr1 所指向的物件刪除後...
 Window A:
-----*/



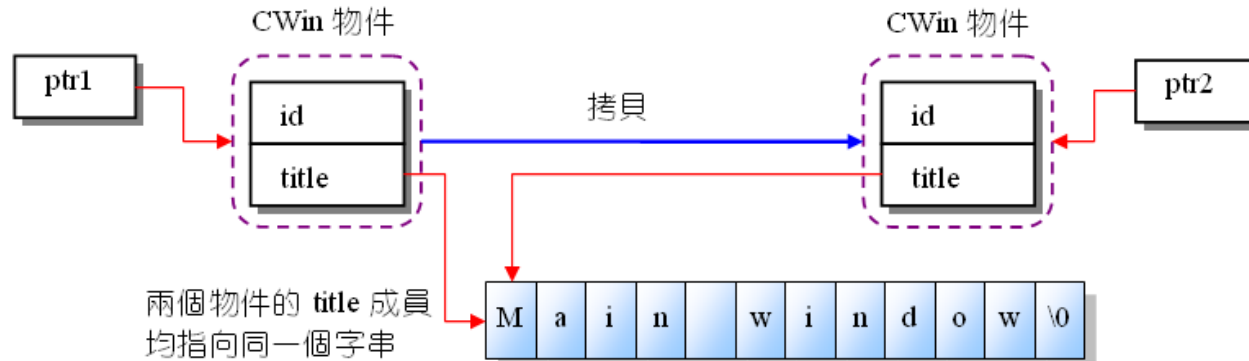
錯誤的使用拷貝建構元 (3/6)

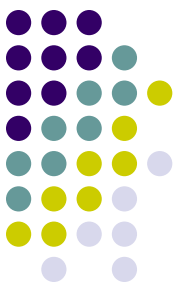
- 下圖為prog14_8記憶體空間配置的情形

```
CWin *ptr1=new CWin('A',"Main window");
```



```
CWin *ptr2=new CWin(*ptr1);
```





錯誤的使用拷貝建構元 (4/6)

- 下面的程式碼修正prog14_8的錯誤

```

01 // prog14_9, 自行撰寫拷貝建構元
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 class CWin
06 {
07     private:
08         char id, *title;
09
10     public:
11         CWin(char i='D', char *text="Default window"):id(i)
12         {
13             cout << "建構元被呼叫了..." << endl;
14             title=new char[strlen(text)+1];           // 配置記憶空間
15             strcpy(title,text);
16         }
17         CWin(const CWin &win)
18         {
19             cout << "拷貝建構元被呼叫了..." << endl;
20             id=win.id;
21             title=new char[strlen(win.title)+1];       // 配置記憶空間
22             strcpy(title,win.title);
23         }

```

/* prog14_9 OUTPUT-----

建構元被呼叫了...
 拷貝建構元被呼叫了...
 Window A: Main window
 Window A: Main window
 將ptr1 所指向的物件刪除後...
 Window A: Main window

-----*/



錯誤的使用拷貝建構元 (5/6)

```

24     ~CWin()                                // 解構元的原型
25     {
26         delete [] title;
27     }
28     void show(void)
29     {
30         cout << "Window " << id << ": " << title << endl;
31     }
32 };
33
34 int main(void)
35 {
36     CWin *ptr1=new CWin('A',"Main window");
37     CWin *ptr2=new CWin(*ptr1);
38
39     ptr1->show();
40     ptr2->show();
41
42     delete ptr1;
43     cout << "將 ptr1 所指向的物件刪除後..." << endl;
44     ptr2->show();
45
46     delete ptr2;
47     system("pause");
48     return 0;
49 }

```

/* prog14_9 OUTPUT-----

建構元被呼叫了...

拷貝建構元被呼叫了...

Window A: Main window

Window A: Main window

將 ptr1 所指向的物件刪除後...

Window A: Main window

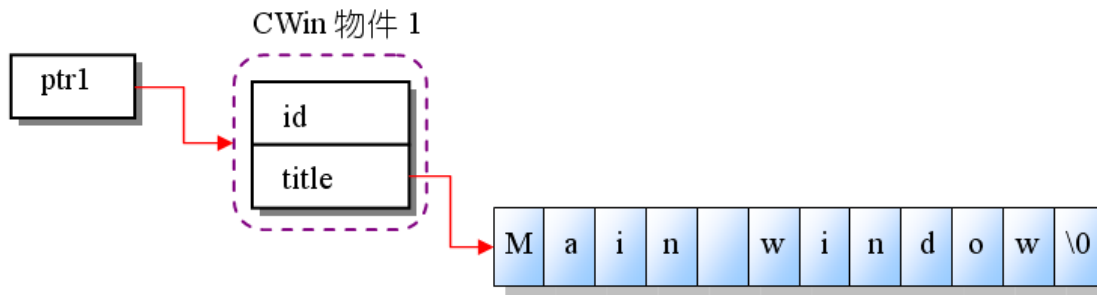
-----*/



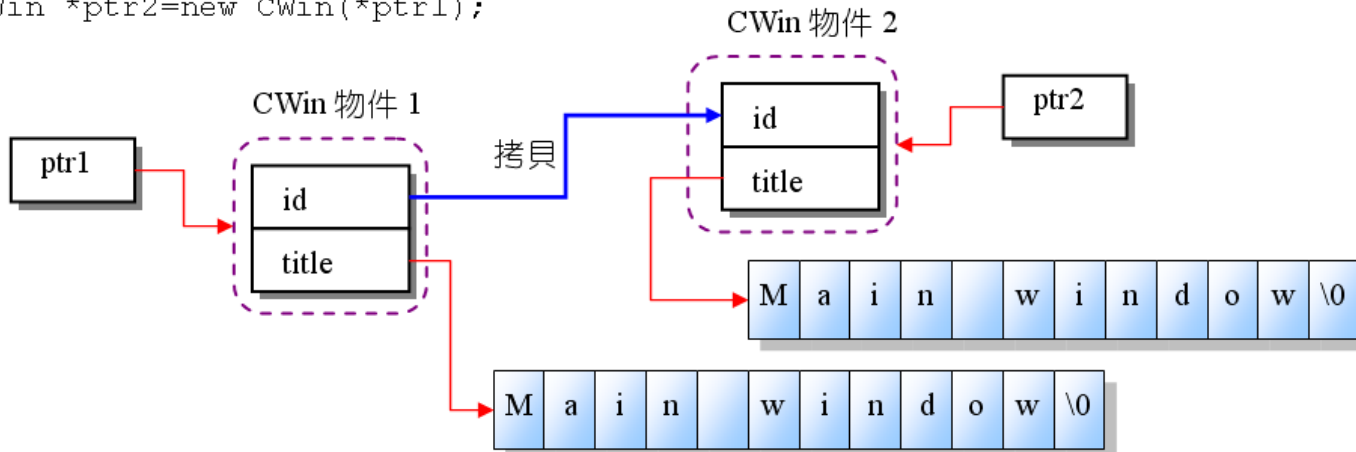
錯誤的使用拷貝建構元 (6/6)

- 下圖為prog14_9執行時，指標與記憶體之配置情形

```
CWin *ptr1=new CWin('A',"Main window");
```



```
CWin *ptr2=new CWin(*ptr1);
```





常犯的錯誤 (1/5)

- 下面的程式是未撰寫拷貝建構元的錯誤示範

```

01  // prog14_10, 錯誤示範，未撰寫拷貝建構元的錯誤
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  class CWin
06  {
07      private:
08          char id, *title;
09
10      public:
11          CWin(char i='D', char *text="Defaule window"):id(i)
12          {
13              cout << "建構元被呼叫了..." << endl;
14              title=new char[strlen(text)+1];      // 配置記憶空間
15              strcpy(title,text);
16          }
17          ~CWin()                                  // 解構元的原型
18          {
19              delete [] title;
20          }

```

/* prog14_10 OUTPUT---

建構元被呼叫了...

Window A: Main window

Window A:

-----*/



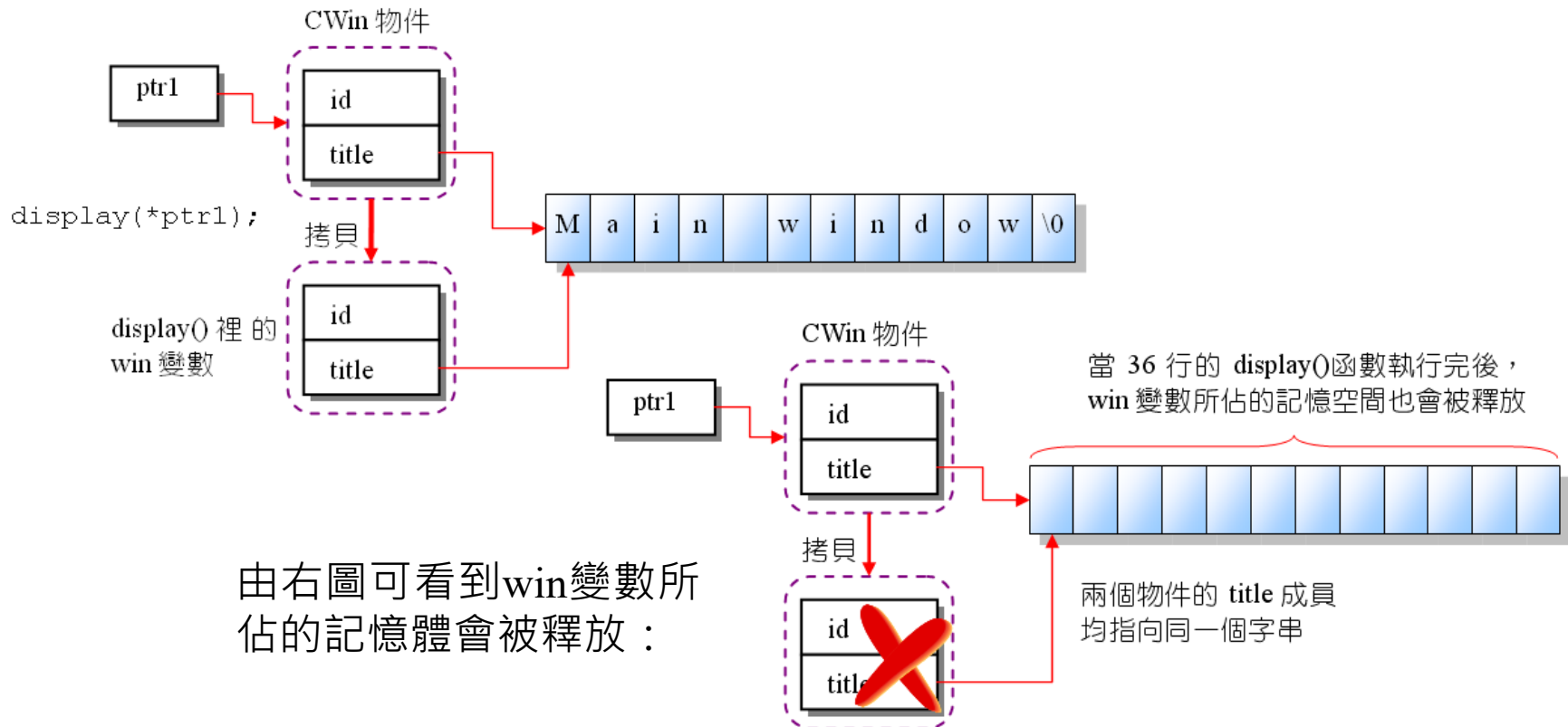
常犯的錯誤 (2/5)

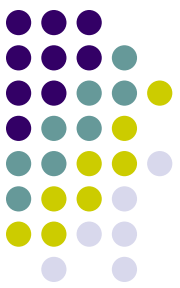
```
21     void show()
22     {
23         cout << "Window " << id << ": " << title << endl;
24     }
25 };
26
27 void display(CWin win)           // 用來呼叫CWin類別裡的 show() 函數
28 {
29     win.show();
30 }
31
32 int main(void)
33 {
34     CWin *ptr1=new CWin('A',"Main window");
35
36     display(*ptr1);
37     display(*ptr1);
38
39     delete ptr1;
40
41     system("pause");
42     return 0;
43 }
```

/* prog14_10 OUTPUT---
建構元被呼叫了...
Window A: Main window
Window A:
-----*/

- 執行36行時，指標與記憶體之配置情形

```
36      display(*ptr1);
```





常犯的錯誤 (4/5)

- 下面的程式碼是修正prog14_10過後的版本

```
01 // prog14_11, 加入拷貝建構元來修正錯誤
```

```
02 #include <iostream>
```

```
03 #include <cstdlib>
```

```
04 using namespace std;
```

```
05 class CWin
```

```
06 {
```

```
07     private:
```

```
08         char id, *title;
```

```
09
```

```
10     public:
```

```
11         CWin(char i='D', char *text="Defaule window"):id(i)
```

```
12         {
```

```
13             cout << "建構元被呼叫了..." << endl;
```

```
14             title=new char[strlen(text)+1]; // 配置記憶空間
```

```
15             strcpy(title,text);
```

```
16         }
```

```
17         CWin(const CWin &win)
```

```
18         {
```

```
19             cout << "拷貝建構元被呼叫了..." << endl;
```

```
20             id=win.id;
```

```
21             title=new char[strlen(win.title)+1]; // 配置記憶空間
```

```
22             strcpy(title,win.title);
```

```
23         }
```

```
/* prog14_11 OUTPUT-----
```

```
建構元被呼叫了...
```

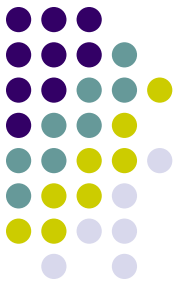
```
拷貝建構元被呼叫了...
```

```
Window A: Main window
```

```
拷貝建構元被呼叫了...
```

```
Window A: Main window
```

```
-----*/
```



常犯的錯誤 (5/5)

```
24     ~CWin()                                // 解構元的原型
25     {
26         delete [] title;
27     }
28     void show()
29     {
30         cout << "Window " << id << ": " << title << endl;
31     }
32 };
33
34 void display(CWin win)
35 {
36     win.show();
37 }
38
39 int main(void)
40 {
41     CWin *ptr1=new CWin('A',"Main window");
42
43     display(*ptr1);
44     display(*ptr1);
45
46     delete ptr1;
47
48     system("pause");
49     return 0;
50 }
```

/* prog14_11 OUTPUT-----

建構元被呼叫了...

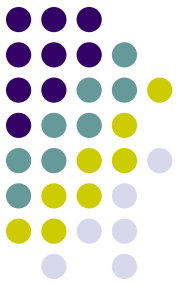
拷貝建構元被呼叫了...

Window A: Main window

拷貝建構元被呼叫了...

Window A: Main window

-----*/



The End-