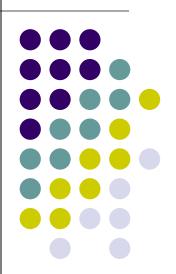
## 第十四章 解構元與拷貝建構元

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

### 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

```
// prog14 1, 解構元的使用
01
02
    #include <iostream>
    #include <cstdlib>
03
    using namespace std;
04
    class CWin
                              // 定義視窗類別 CWin
0.5
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
           cout << "建構元被呼叫了..." << endl;
14
15
16
        ~CWin()
                              // 解構元
17
           cout << "解構元被呼叫了, Win " << this->id << "被銷毁了..." << endl;
18
19
           system("pause");
20
```



### 解構元的使用 (2/2)

```
void show member(void)
21
22
23
           cout << "Window " << id << ": ";
24
           cout << "width=" << width << ", height=" << height << endl;</pre>
25
26
    };
27
                                /* prog14 1 OUTPUT-----
28
    int main(void)
                                建構元被呼叫了...
29
                                建構元被呼叫了...
30
       CWin win1('A',50,40);
                                建構元被呼叫了...
       CWin win2('B', 40,50);
31
                                建構元被呼叫了...
                                Window A: width=50, height=40
32
       CWin win3('C', 60, 70);
                                Window B: width=40, height=50
33
       CWin win4('D',90,40);
                                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
34
                                請按任意鍵繼續 · · · · ———— 執行第 19 行的結果
35
       win1.show member();
                                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
36
       win2.show member();
                                               ————— 執行第 19 行的結果
                                請按仟意鍵繼續 . . .
                                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
37
                                請按仟意鍵繼續 . . . — — — 執行第 19 行的結果
38
       system("pause");
                                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
39
       return 0;
40
```





#### 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)



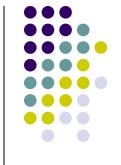
先看一個簡單的例子,此例無關動態記憶體配置

```
01
    // prog14 2, 固定空間的記憶體配置
02
    #include <iostream>
03
    #include <cstdlib>
    using namespace std;
04
    class CWin
                                  // 定義視窗類別 CWin
05
06
07
      private:
        char id, title[20];
08
09
      public:
10
        CWin(char i='D', char *text="Default window"):id(i)
11
12
           strcpy(title,text); // 將 text 指向的字串拷貝到 title 陣列裡
13
14
        ~CWin()
                                  // 解構元
15
16
           cout << "解構元被呼叫了,Win " << this->id << "被銷毀了.." << endl;、
17
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
                                               /* prog14 2 OUTPUT-----
24
    };
25
                                               Window A: Main window
    int main(void)
26
                                               Window B: Default window
27
                                               sizeof(win1) = 21
                                               sizeof(win2) = 21
28
       CWin win1('A', "Main window");
                                               請按任意鍵繼續 . . .
       CWin win2('B');
29
                                               解構元被呼叫了, Win B被銷毀了...
30
                                               請按任意鍵繼續 . . .
31
       win1.show();
                                               解構元被呼叫了, Win A被銷毀了...
32
       win2.show();
                                               請按任意鍵繼續 . . .
33
34
       cout << "sizeof(win1) = " << sizeof(win1) << endl;</pre>
35
       cout << "sizeof(win2) = " << sizeof(win2) << endl;</pre>
36
       system("pause");
37
38
       return 0;
                                                                           7
39
```

#### 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

```
// prog14 3, 使用動態記憶體配置
01
   #include <iostream>
02
   #include <cstdlib>
03
   using namespace std;
   class CWin
                                     // 定義視窗類別 CWin
05
06
      private:
07
        char id, *title;
                                     // 宣告 title 為指向字元陣列的指標
08
09
      public:
10
        CWin(char i='D', char *text="Default window"):id(i)
11
12
13
          title=new char[strlen(text)+1];
                                                  // 配置記憶體空間
14
          strcpy(title,text);
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;
                                     // 釋放 title 所指向的記憶體空間
          delete [] title;
19
          system("pause");
20
21
```

### 使用動態記憶體配置 (2/3)



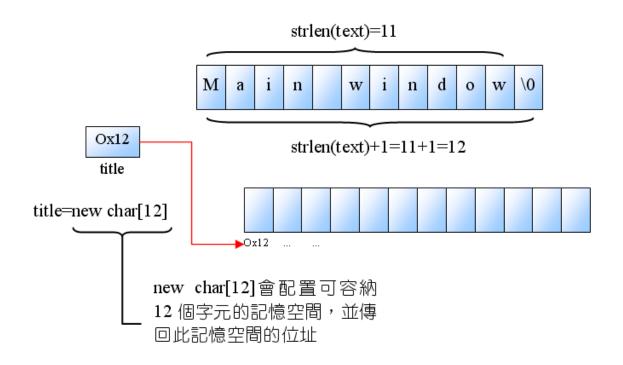
• 原始程式編譯及連結的過程

```
22
         void show(void)
23
            cout << "Window " << id << ": " << title << endl;</pre>
24
25
                                                          /* prog14 3 OUTPUT-----
26
    };
                                                          Window A: Main window
27
                                                          Window B: Default window
28
    int main(void)
                                                          sizeof(win1) = 8
29
                                                          sizeof(win2) = 8
                                                          請按任意鍵繼續 . . .
       CWin win1('A', "Main window");
30
                                                          解構元被呼叫了, Win B被銷毀了...
       CWin win2('B');
31
                                                          請按任意鍵繼續 . . .
32
                                                          解構元被呼叫了, Win A被銷毀了...
33
       win1.show();
                                                          請按任意鍵繼續 . . .
       win2.show();
34
35
       cout << "sizeof(win1) = " << sizeof(win1) << endl;</pre>
36
       cout << "sizeof(win2) = " << sizeof(win2) << endl;</pre>
37
       system("pause");
38
                                                                                    9
39
       return 0;
```

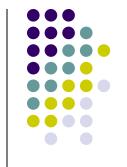
### 使用動態記憶體配置 (3/3)



• 下圖為prog14 3中,記憶空間配置過程的說明



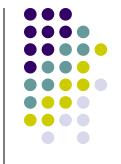
### 錯誤的使用動態記憶體配置 (1/2)



● 下面的範例修改自prog14\_3, 這是個錯誤的示範

```
// prog14 4, 使用動態記憶體配置, 錯誤的示範
01
    #include <iostream>
02
                                            /* prog14 4 OUTPUT-----
    #include <cstdlib>
03
    using namespace std;
04
                                            Window A: Main window
05
                                            Window B: Defaule window
06
    // 將 progl4 3 CWin 類別的定義放在這裡
                                            請按任意鍵繼續 . . .
                                            解構元被呼叫了, Win A被銷毀了...
07
                                            請按任意鍵繼續 . . .
08
    int main(void)
09
10
      CWin win1('A', "Main window");
                                     // 宣告ptr 為指向 CWin 物件的指標
11
      CWin *ptr;
                                     // 建立新的物件,並讓 ptr 指向它
12
      ptr=new CWin('B');
13
                                     // 以 win1 物件呼叫 show()函數
14
      win1.show();
      ptr->show();
                                     // 以 ptr 指標呼叫 show() 函數
15
16
      system("pause");
17
                                                                     11
18
      return 0;
19
```

### 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

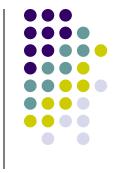
```
// prog14 5, 使用動態記憶體配置
01
    #include <iostream>
    #include <cstdlib>
0.3
    using namespace std;
04
0.5
06
    // 將 progl4 3 CWin 類別的定義放在這裡
07
08
    int main(void)
                                            /* prog14 5 OUTPUT-----
09
                                            Window A: Main window
      CWin win1('A', "Main window");
10
                                            Window B: Defaule window
11
      CWin *ptr;
                                            請按任意鍵繼續 . . .
      ptr=new CWin('B');
12
                                            解構元被呼叫了, Win B被銷毀了...
1.3
                                            請按任意鍵繼續 . . .
      win1.show();
14
                                            解構元被呼叫了, Win A被銷毀了...
15
      ptr->show();
                                            請按任意鍵繼續 . . .
16
      system("pause");
17
18
                                    // 釋放 ptr 所指向物件之記憶體
19
      delete ptr;
20
21
      return 0;
```

#### 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

```
// prog14 6, 預設的拷貝建構元
    #include <iostream>
02
    #include <cstdlib>
03
    using namespace std;
05
    class CWin
                                            // 定義視窗類別 CWin
06
                                                         /* prog14 6 OUTPUT-----
      private:
07
        char id;
                                                         建構元被呼叫了...
08
                                                         Window A: width=50, height=40
09
        int width, height;
                                                         Window A: width=50, height=40
10
      public:
11
12
        CWin(char i,int w,int h):id(i),width(w),height(h)
13
           cout << "建構元被呼叫了..." <<endl;
14
15
16
        void show member (void)
17
           cout << "Window " << id << ": ";
18
           cout << "width=" << width << ", height=" << height << endl;</pre>
19
                                                                                 14
20
21
    };
```

#### 預設的拷貝建構元 (3/3)



```
建構元被呼叫了...
                                                       Window A: width=50, height=40
22
                                                       Window A: width=50, height=40
    int main(void)
23
24
25
                                            // 建立 win1 物件
       CWin win1('A',50,40);
26
       CWin win2 (win1);
                                               呼叫預設的拷貝建構元
27
28
       win1.show member();
      win2.show member();
29
30
```

- 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:
```

system("pause");

return 0;

31

32

```
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

#### 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

```
// prog14 7, 撰寫自己的拷貝建構元
01
                                                        /* prog14 7 OUTPUT-----
    #include <iostream>
02
    #include <cstdlib>
0.3
                                                        建構元被呼叫了...
    using namespace std;
04
                                                        拷貝建構元被呼叫了...
    class CWin
                                     // 定義視窗類別 CWin
05
                                                        Window A: width=50, height=40
06
                                                        Window A: width=50, height=40
      private:
07
        char id;
08
        int width, height;
09
10
      public:
11
12
        CWin(char i, int w, int h):id(i), width(w), height(h)
13
          cout << "建構元被呼叫了..." << endl;
14
15
                                    // 定義拷貝建構元
16
        CWin (const CWin &win)
17
          cout << "拷貝建構元被呼叫了..." << endl;
18
          id=win.id;
19
          width=win.width;
                               拷貝資料成員
20
          height=win.height;
21
22
```

### 撰寫自己的拷貝建構元 (3/3)

```
void show member (void)
23
24
           cout << "Window " << id << ": ";
25
           cout << "width=" << width << ", height=" << height << endl;</pre>
26
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();
                                    /* prog14 7 OUTPUT-----
36
       win2.show member();
37
                                    建構元被呼叫了...
38
       system("pause");
                                    拷貝建構元被呼叫了...
39
       return 0;
                                    Window A: width=50, height=40
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

```
// prog14 8, 錯誤示範,未撰寫拷貝建構元而使用預設的版本
    #include <iostream >
   #include <cstdlib>
    using namespace std;
    class CWin
                                             // 定義視窗類別 CWin
0.5
06
07
      private:
        char id, *title;
08
09
10
      public:
        CWin(char i='D', char *text="Default window"):id(i)
11
12
          cout << "建構元被呼叫了..." << endl;
13
14
          title=new char[strlen(text)+1]; // 配置記憶空間
15
          strcpy(title,text);
16
        CWin (const CWin &win)
17
18
19
          cout << "拷貝建構元被呼叫了..." <<endl;
20
          id=win.id;
                             拷貝資料成員
21
          title=win.title;
22
```

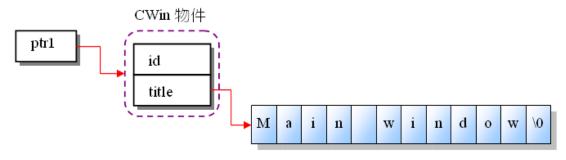
### 錯誤的使用拷貝建構元 (2/6)

```
// 解構元的原型
23
        ~CWin()
24
25
          delete [] title;
26
27
        void show(void)
28
          cout << "Window " << id << ": " << title << endl;</pre>
29
30
31
    };
32
33
    int main(void)
34
35
      CWin *ptrl=new CWin('A', "Main window");
      CWin *ptr2=new CWin(*ptr1); // 以ptr1 所指向的物件為初值建立新物件
36
37
                                                        /* prog14 8 OUTPUT-----
      ptrl->show();
38
                                                        建構元被呼叫了...
39
      ptr2->show();
40
                                                        拷貝建構元被呼叫了...
                                // 釋放 ptrl 所指向的記憶空間 Window A: Main window
41
      delete ptrl;
      cout << "將 ptrl 所指向的物件刪除後..." << endl;
42
                                                        Window A: Main window
43
      ptr2->show();
                                                        | 將 ptr1 所指向的物件刪除後...
44
                                                        Window A:
45
                                // 釋放 ptr2 所指向的記憶空間
      delete ptr2;
      system("pause");
46
                                                                                   20
      return 0;
47
```

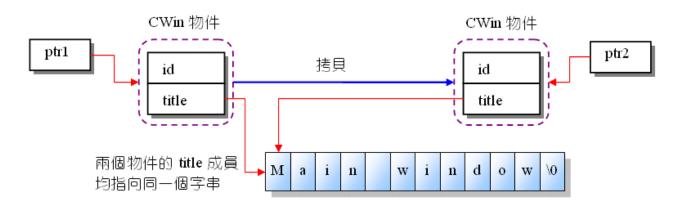
### 錯誤的使用拷貝建構元 (3/6)

• 下圖為prog14 8記憶體空間配置的情形

CWin \*ptrl=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

```
// prog14 9, 自行撰寫拷貝建構元
01
                                               /* prog14 9 OUTPUT-----
    #include <iostream>
02
0.3
    #include <cstdlib>
                                               建構元被呼叫了...
    using namespace std;
                                               拷貝建構元被呼叫了...
    class CWin
05
                                               Window A: Main window
06
                                               Window A: Main window
07
      private:
                                               將 ptrl 所指向的物件刪除後...
        char id, *title;
08
                                               Window A: Main window
09
      public:
10
        CWin(char i='D', char *text="Default window"):id(i)
11
12
          cout << "建構元被呼叫了..." << endl;
1.3
                                        // 配置記憶空間
14
          title=new char[strlen(text)+1];
15
          strcpy(title,text);
16
17
        CWin(const CWin &win)
18
          cout << "拷貝建構元被呼叫了..." << endl;
19
20
          id=win.id:
21
          title=new char[strlen(win.title)+1]; // 配置記憶空間
22
          strcpy(title, win.title);
23
```

### 錯誤的使用拷貝建構元 (5/6)

```
~CWin()
24
                                                   // 解構元的原型
25
26
           delete [] title;
27
28
        void show(void)
29
           cout << "Window " << id << ": " << title << endl;</pre>
30
31
32
    };
33
34
    int main(void)
                                                      /* prog14 9 OUTPUT-----
35
      CWin *ptrl=new CWin('A', "Main window");
36
                                                      建構元被呼叫了...
37
      CWin *ptr2=new CWin(*ptr1);
                                                      拷貝建構元被呼叫了...
38
                                                      Window A: Main window
39
      ptr1->show();
40
      ptr2->show();
                                                      Window A: Main window
41
                                                      將 ptrl 所指向的物件刪除後...
42
      delete ptr1;
                                                      Window A: Main window
      cout << "將 ptrl 所指向的物件刪除後..." << endl;
43
      ptr2->show();
44
45
46
      delete ptr2;
      system("pause");
47
                                                                                  23
       return 0;
48
```

### 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 \*ptrl=new CWin('A', "Main window"); CWin 物件 1 ptr1 title \0 i d n w n w CWin \*ptr2=new CWin(\*ptr1); CWin 物件 2 CWin 物件 1 ptr2 id 拷貝 ptr1 title id title  $\mathbf{M}$ n w n a

n

w

d

n

\0

w

 $\mathbf{M}$ 



#### 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

```
// prog14 10, 錯誤示範,未撰寫拷貝建構元的錯誤
01
    #include <iostream>
02
    #include <cstdlib>
0.3
                                           /* prog14 10 OUTPUT---
    using namespace std;
04
    class CWin
0.5
                                           建構元被呼叫了...
06
                                           Window A: Main window
      private:
07
                                           Window A:
08
        char id, *title;
09
      public:
10
        CWin(char i='D', char *text="Defaule window"):id(i)
11
12
13
          cout << "建構元被呼叫了..." << endl;
          title=new char[strlen(text)+1]; // 配置記憶空間
14
15
          strcpy(title,text);
16
        ~CWin()
                                             // 解構元的原型
17
18
          delete [] title;
19
20
```



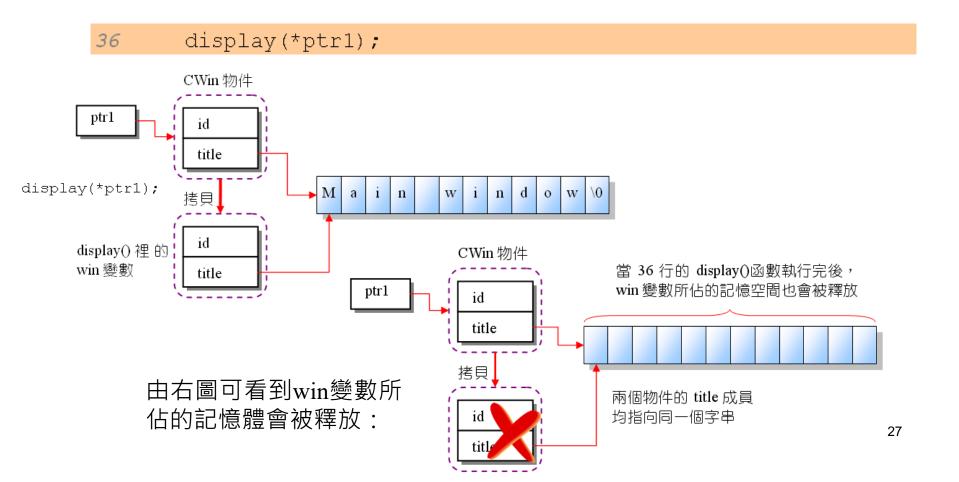
#### 常犯的錯誤 (2/5)

```
void show()
21
22
           cout << "Window " << id << ": " << title << endl;</pre>
23
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
      CWin *ptr1=new CWin('A', "Main window");
34
35
36
      display(*ptr1);
37
      display(*ptr1);
38
                                                /* prog14_10 OUTPUT---
39
      delete ptr1;
                                                建構元被呼叫了...
40
                                                Window A: Main window
      system("pause");
41
42
      return 0:
                                                Window A:
43
```



#### 常犯的錯誤 (3/5)

• 執行36行時,指標與記憶體之配置情形





#### 常犯的錯誤 (4/5)

• 下面的程式碼是修正prog14\_10過後的版本

```
// prog14 11, 加入拷貝建構元來修正錯誤
                                            /* prog14 11 OUTPUT-----
   #include <iostream>
02
   #include <cstdlib>
0.3
                                            建構元被呼叫了...
   using namespace std;
04
                                            拷貝建構元被呼叫了...
   class CWin
05
                                            Window A: Main window
06
                                            拷貝建構元被呼叫了...
      private:
07
        char id, *title;
                                            Window A: Main window
08
09
10
      public:
        CWin(char i='D', char *text="Defaule window"):id(i)
11
12
          cout << "建構元被呼叫了..." << endl;
13
          title=new char[strlen(text)+1]; // 配置記憶空間
14
15
          strcpy(title,text);
16
       CWin (const CWin &win)
17
18
          cout << "拷貝建構元被呼叫了..." << endl;
19
20
          id=win.id;
21
          title=new char[strlen(win.title)+1]; // 配置記憶空間
22
          strcpy(title,win.title);
23
```



#### 常犯的錯誤 (5/5)

```
// 解構元的原型
        ~CWin()
24
25
26
          delete [] title;
27
        void show()
28
29
           cout << "Window " << id << ": " << title << endl;</pre>
30
31
32
    };
33
34
    void display (CWin win)
35
                                                 /* prog14 11 OUTPUT-----
36
      win.show();
37
                                                 建構元被呼叫了...
38
                                                 拷貝建構元被呼叫了...
39
    int main(void)
                                                 Window A: Main window
40
                                                 拷貝建構元被呼叫了...
41
      CWin *ptrl=new CWin('A', "Main window");
                                                 Window A: Main window
42
43
      display(*ptr1);
44
      display(*ptr1);
45
46
      delete ptr1;
47
       system("pause");
48
       return 0;
49
```



# -The End-