

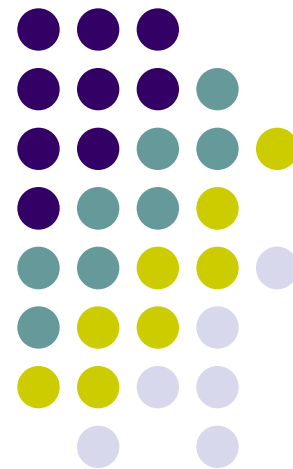
類別的繼承

認識繼承的概念

學習由子類別存取父類別的成員

熟悉改寫的技術

在子類別中使用拷貝建構元





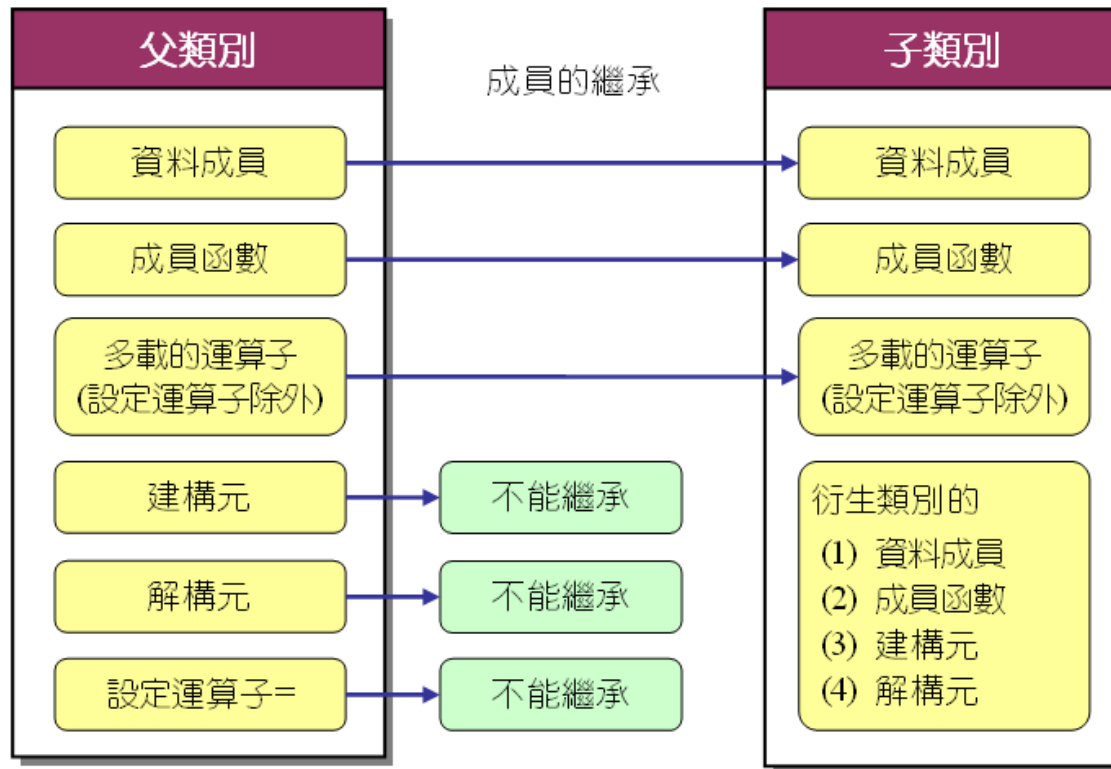
基底類別與衍生類別 (1/2)

- 以既有類別為基礎，進而衍生出另一類別，稱為「類別的繼承」（inheritance of classes）
- 原有的類別稱為「父類別」（super class）或「基底類別」（basis class）
- 因繼承而產生的新類別則稱為「子類別」（sub class）或「衍生類別」（derived class）



基底類別與衍生類別 (2/2)

- 類別成員繼承的關係





簡單的繼承範例 (1/6)

- 類別繼承的格式

```
class 父類別名稱    // 定義父類別  
{  
    父類別裡的各種成員  
}
```

} 父類別

```
class 子類別名稱 : 修飾子 父類別名稱  
{  
    子類別裡的各種成員  
}
```

可為 public, private 或 protected

} 子類別，即由父類別衍生而出的類別



簡單的繼承範例 (2/6)

- 下面的範例簡單說明繼承的使用方法

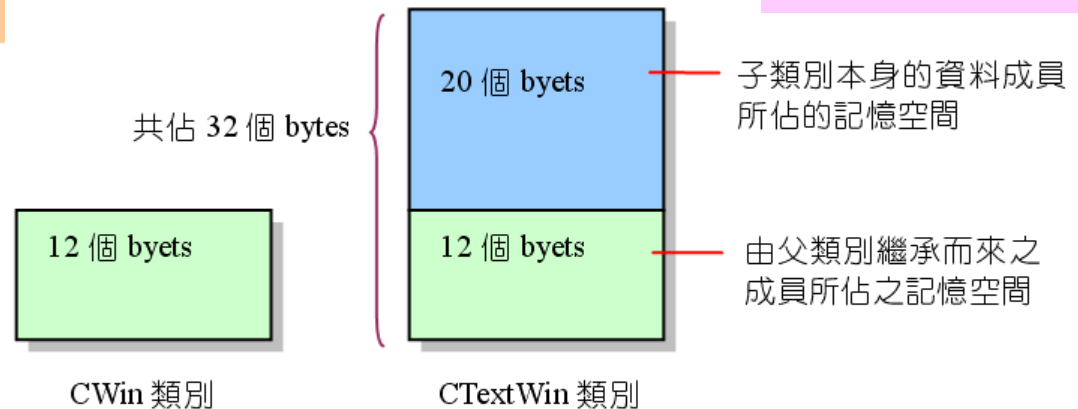
```
01 // prog16_1, 繼承的簡單範例
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 class CWin // 定義 CWin 類別，在此為父類別
06 {
07     private:
08         char id;
09         int width,height;
10
11     public:
12         CWin(char i='D',int w=10,int h=10):id(i),width(w),height(h)
13         {
14             cout << "CWin()建構元被呼叫了..." << endl;
15         }
16         void show member(void) // 成員函數，用來顯示資料成員的值
17         {
18             cout << "Window " << id << ": ";
19             cout << "width=" << width << ", height=" << height << endl;
20         }
21 };
```

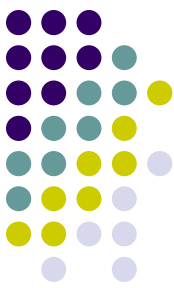


簡單的繼承範例 (3/6)

```
22
23 class CTextWin : public CWin // 定義 CTextWin 類別，繼承自 CWin 類別
24 {
25     private:                                // 子類別裡的私有成員
26         char text[20];
27
28     public:                                // 子類別裡的公有成員
29         CTextWin(char *tx)                // 子類別的建構元
30         {
31             cout << "CTextWin() 建構元被呼叫了..." << endl;
32             strcpy(text, tx);
33         }
34         void show text()                  // 子類別的成員函數
35         {
36             cout << "text = " << text << endl;
37         }
38     };
39
```

下圖是本例中，父類別與子類別所佔記憶體之比較





簡單的繼承範例 (4/6)

```

40  int main(void)
41  {
42      CWin win('A',50,60);           // 建立父類別的物件
43      CTextWin txt("Hello C++");   // 建立子類別的物件
44
45      win.show member();             // 以父類別物件呼叫父類別的函數
46      txt.show member();            // 以子類別物件呼叫父類別的函數
47      txt.show text();              // 以子類別物件呼叫子類別的函數
48
49      cout << "win 物件佔了 " << sizeof(win) << " bytes" << endl;
50      cout << "txt 物件佔了 " << sizeof(txt) << " bytes" << endl;
51
52      system("pause");
53      return 0;
54  }
```

/* prog16_1 OUTPUT-----

CWin() 建構元被呼叫了... ——— 42 行建立父類別的物件所得的結果

CWin() 建構元被呼叫了...

CTextWin() 建構元被呼叫了...

} 43 行會先呼叫父類別的建構元，再呼叫子類別的建構元

Window A: width=50, height=60

——— 45 行以 win 物件呼叫 show_member()

Window D: width=10, height=10

——— 46 行以 txt 物件呼叫 show_member()

text = Hello C++

——— 47 行以 txt 物件呼叫 show_text()

win 物件佔了 12 bytes

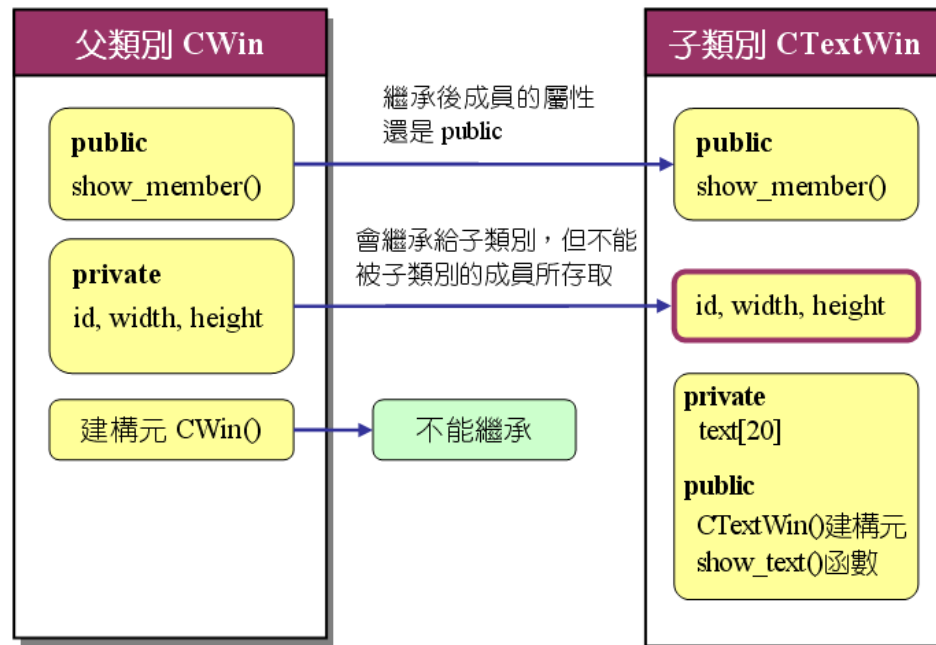
txt 物件佔了 32 bytes

-----*/



簡單的繼承範例 (5/6)

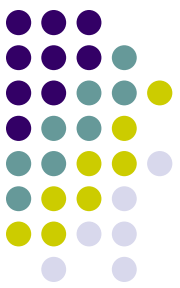
- 本例的繼承關係圖繪製如下





簡單的繼承範例 (6/6)

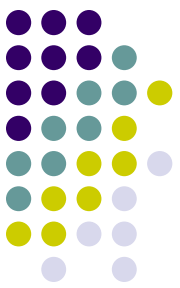
- 由前一個範例可學到下列幾點重要的觀念
 - 透過類別的繼承，可將父類別的成員繼承給子類別
 - 在執行子類別的建構元之前，會先自動呼叫父類別中沒有引數的建構元
 - 子類別物件所佔的位元組，等於自己資料成員所佔的位元組，加上繼承過來之成員所佔用的位元組



呼叫父類別中特定的建構元 (1/4)

- 下面是呼叫父類別CWin裡特定建構元的範例

```
01 // prog16_2, 設定運算子多載的進階應用
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='D',int w=10,int h=10):id(i),width(w),height(h)
13         {
14             cout << "CWin()建構元被呼叫了..." << endl;
15         }
16         CWin(int w,int h):width(w),height(h)
17         {
18             cout << "CWin(int w,int h)建構元被呼叫了..." << endl;
19             id='K';
20         }
```



呼叫父類別中特定的建構元 (2/4)

```
21     void show member(void)
22     {
23         cout << "Window " << id << ": ";
24         cout << "width=" << width << ", height=" << height << endl;
25     }
26 };
27
28 class CTextWin : public CWin
29 {
30     private:
31         char text[20];
32
33     public:
34         CTextWin(int w,int h) : CWin(w,h)    // CTextWin(int,int)建構元
35         {
36             cout << "CTextWin(int w,int h)建構元被呼叫了..." << endl;
37             strcpy(text,"Have a good night");
38         }
39         CTextWin(char *tx)    // CTextWin(char *)
40         {
41             cout << "CTextWin(char *tx)建構元被呼叫了..." << endl;
42             strcpy(text,tx);
43         }
```

呼叫父類別裡 16~20 行的
CWin(w,h) 建構元



呼叫父類別中特定的建構元 (3/4)

```

44     void show text()
45     {
46         cout << "text = " << text << endl;
47     }
48 };
49
50 int main(void)
51 {
52     CTextWin tx1("Hello C++"); // 呼叫 39~43 行的 CTextWin() 建構元
53     CTextWin tx2(60,70);       // 呼叫 34~38 行的 CTextWin() 建構元
54
55     tx1.show member();
56     tx1.show text();
57
58     tx2.show member();
59     tx2.show text();
60
61     system("pause");
62     return 0;
63 }

```

/* prog16_2 OUTPUT-----

CWin() 建構元被呼叫了...

CTextWin(char *tx) 建構元被呼叫了...

CWin(int w,int h) 建構元被呼叫了...

CTextWin(int w,int h) 建構元被呼叫了...

Window D: width=10, height=10

text = Hello C++

Window K: width=60, height=70

text = Have a good night

-----*/

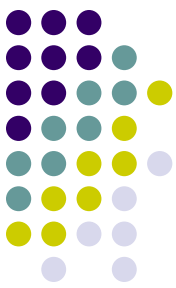
} 52 行建立 tx1 物件後的執行結果

} 53 行建立 tx2 物件後的執行結果



呼叫父類別中特定的建構元 (4/4)

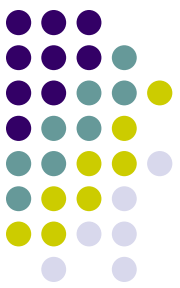
- 這裡有很重要的兩點要提醒讀者：
 - 如果省略34行的敘述，則父類別中沒有引數的建構元還是會被呼叫。
 - 呼叫父類別中特定的建構元，其敘述必須寫在子類別建構元第一行的後面，並以「：」連接，不能置於它處



使用建構元常見的錯誤 (1/3)

- 下面是呼叫父類別建構元時常犯的錯誤範例

```
01 // prog16_3, 呼叫父類別建構元時常犯的錯誤
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(int w,int h):width(w),height(h) // 有兩個引數的建構元
13         {
14             cout << "CWin(int w,int h)建構元被呼叫了..." << endl;
15             id='K';
16         }
17 };
18
```

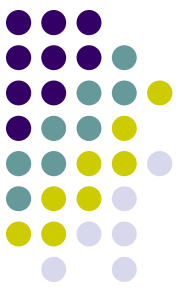


使用建構元常見的錯誤 (2/3)

```
19 class CTextWin : public CWin // 定義子類別 CTextWin
20 {
21     private:
22         char text[20];
23
24     public:
25         CTextWin(char *tx) ——— 執行此建構元之前，會先呼叫父類別
26         {                               裡沒有引數的建構元
27             cout << "CTextWin() 建構元被呼叫了..." << endl;
28             strcpy(text, tx);
29         }
30 };
31
32 int main(void)
33 {
34     CTextWin tx1("Hello C++");
35
36     system("pause");
37     return 0;
38 }
```

以 Dev C++ 為例，編譯時會得到如下的錯誤訊息：

no matching function for call to 'CWin::CWin()'



使用建構元常見的錯誤 (3/3)

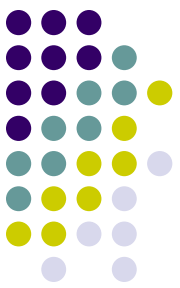
- 下面的程式是修正 prog16_3 的錯誤

```
01 // prog16_4, prog16_3 錯誤的修正
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(int w,int h):width(w),height(h)
13         {
14             cout << "CWin(int w,int h)建構元被呼叫了..." << endl;
15             id='K';
16         }
17         CWin()
18         {
19             cout<< "沒有引數的 CWin() 建構元被呼叫了..." << endl;
20         }
21 };
22
23 // 將 prog16_3, CTextWin 類別的定義放在這兒
24 // 將 prog16_3, main() 主函數放在這兒
```

/* prog16_4 OUTPUT-----

沒有引數的 CWin() 建構元被呼叫了...
CTextWin() 建構元被呼叫了...

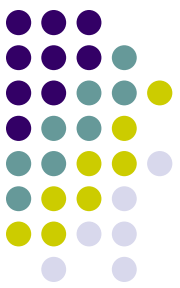
-----*/



父類別裡私有成員的存取 (1/4)

- 錯誤的例子--存取到父類別裡的私有成員

```
01 // prog16_5, 錯誤的例子--存取到父類別裡的私有成員
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 class CWin // 定義父類別 CWin
06 {
07     private:
08         char id;
09
10     public:
11         CWin(char i):id(i) {}
12 };
13
14 class CTextWin : public CWin // 定義子類別 CTextWin
15 {
16     private:
17         char text[20];
18
```



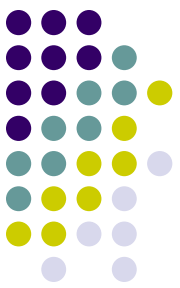
父類別裡私有成員的存取 (2/4)

```
19     public:
20         CTextWin(char i, char *tx):CWin(i)
21         {
22             strcpy(text,tx);
23         }
24         void show()
25         {
26             cout << "Window " << id << ": "; // 讀取父類別裡的私有成員
27             cout << "text = " << text << endl;
28         }
29     };
30
31 int main(void)
32 {
33     CTextWin txt('A',"Hello C++");
34
35     txt.show();
36
37     system("pause");
38     return 0;
39 }
```

編譯時將出現下列的錯誤訊息：

'id' : cannot access private
member declared in class 'CWin'

只要在父類別裡建立公有函數來存取它們
即可改正錯誤。



父類別裡私有成員的存取 (3/4)

- 修正prog16_5的錯誤

```
01 // prog16_6, prog16_5 的修正版
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 class CWin // 定義父類別 CWin
06 {
07     private:
08         char id;
09
10     public:
11         CWin(char i):id(i) {}
12
13         char get id() // get id() 函數，用來取得 id 成員
14         {
15             return id;
16         }
17 };
18
```

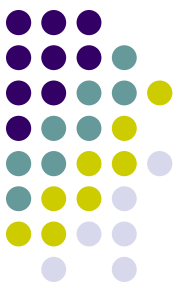
/* prog16_6 OUTPUT-----
Window A: text = Hello C++
-----*/



父類別裡私有成員的存取 (4/4)

```
19 class CTextWin : public CWin          // 定義子類別 CTextWin
20 {
21     private:
22         char text[20];
23
24     public:
25         CTextWin(char i, char *tx):CWin(i)
26         {
27             strcpy(text, tx);
28         }
29         void show()
30         {
31             cout << "Window " << get_id() << ": "; // 呼叫父類別裡的 get_id()
32             cout << "text = " << text << endl;
33         }
34 };
35
36 int main(void)
37 {
38     CTextWin txt('A', "Hello C++");
39
40     txt.show();
41
42     system("pause");
43     return 0;
44 }
```

```
/* prog16_6 OUTPUT-----
Window A: text = Hello C++
-----*/
```

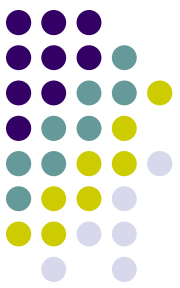


使用protected成員 (1/2)

- 把成員宣告成protected最大的好處是兼顧到成員的安全性與便利性
- 下面的範例是prog16_5的小改版

```
01 // prog16_7, protected 成員的使用
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 class CWin    // 定義視窗類別 CWin
06 {
07     protected:
08         char id;    // 把 id 宣告成 protected 成員，使得它也可以在子類別裡使用
09
10     public:
11         CWin(char i):id(i) {}
12 };
13
```

```
/* prog16_7 OUTPUT-----
Window A: text = Hello C++
-----*/
```



使用protected成員 (2/2)

```
14 class CTextWin : public CWin
15 {
16     private:
17         char text[20];
18
19     public:
20         CTextWin(char i, char *tx):CWin(i)
21         {
22             strcpy(text, tx);
23         }
24         void show()
25         {
26             cout << "Window " << id << ": "; // 讀取父類別裡的保護成員
27             cout << "text = " << text << endl;
28         }
29 };
30
31 // 將prog16 5的main()放在這兒
```

/* prog16_7 OUTPUT-----

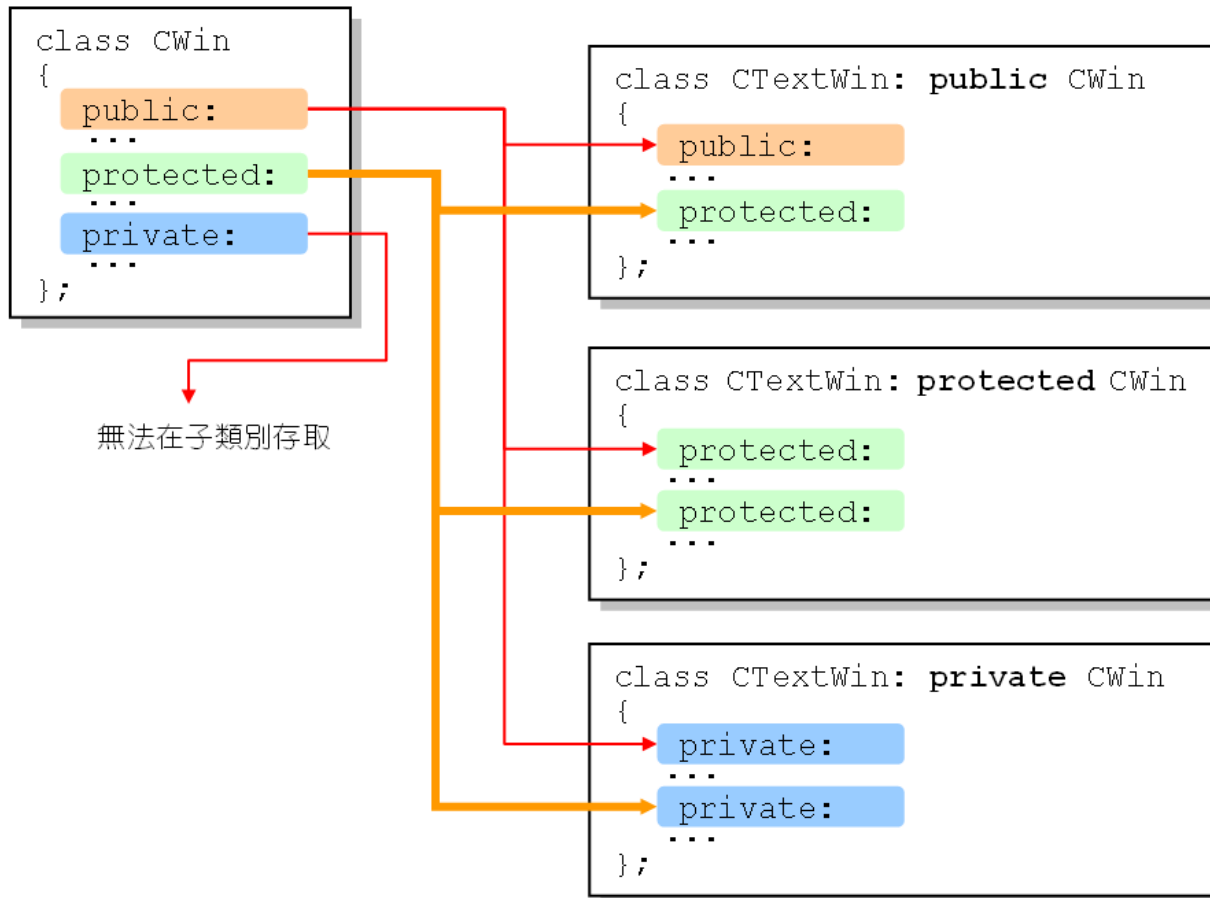
Window A: text = Hello C++

-----*/



類別繼承的存取模式

- 下圖說明類別繼承的存取模式





多載與改寫

- 「多載」是函數名稱相同，在不同的場合可做不同的事
- 「改寫」是在子類別裡定義與父類別名稱相同的函數，用來覆蓋父類別裡函數功能的一種技術

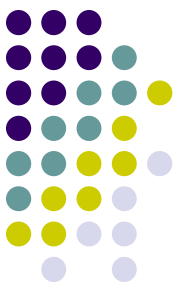


改寫的範例 (1/2)

- 簡單的改寫範例

```
01 // prog16_8, 繼承的簡單範例
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 class CWin // 定義 CWin 類別，在此為父類別
06 {
07     protected:
08         char id;
09
10     public:
11         CWin(char i):id(i){}
12
13         void show member(void) // 父類別的 show member() 函數
14         {
15             cout << "父類別的 show member() 函數被呼叫了..." << endl;
16             cout << "Window " << id << endl;
17         }
18 };
19
```

```
/* prog16_8 OUTPUT-----
子類別的 show member() 函數被呼叫了...
Window i: text = Hello C++
-----*/
```



改寫的範例 (2/2)

```

20 class CTextWin : public CWin // 定義 CTextWin 類別，繼承自 CWin 類別
21 {
22     private:                                // 子類別裡的私有成員
23         char text[20];
24
25     public:
26         CTextWin(char i, char *tx):CWin('i') // 子類別的建構元
27         {
28             strcpy(text, tx);
29         }
30         void show member() // 子類別的 show member() 函數
31         {
32             cout << "子類別的 show member() 函數被呼叫了..." << endl;
33             cout << "Window " << id << ": ";
34             cout << "text = " << text << endl;
35         }
36 };
37
38 int main(void)
39 {
40     CTextWin txt('A', "Hello C++"); // 建立子類別的物件
41
42     txt.show member(); // 以子類別物件呼叫 show member() 函數
43
44     system("pause");
45     return 0;
46 }

```

```

/* prog16_8 OUTPUT-----
子類別的 show member() 函數被呼叫了...
Window i: text = Hello C++
-----*/

```



「改寫」與「多載」的比較

- 「多載」：英文名稱為overloading
 - 它是在相同類別內，定義名稱相同，但引數個數或型態不同的函數，C++可依據引數的個數或型態，呼叫相對應的函數
- 「改寫」：英文名稱為overriding
 - 它是在子類別當中，定義名稱、引數個數與型態均與父類別相同的函數，用以改寫父類別裡函數的功用



錯誤的使用拷貝建構元 (1/6)

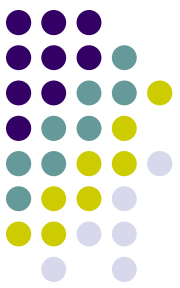
- 如果父類別或子類別裡沒有提供拷貝建構元的話，編譯器會提供一個預設的拷貝建構元
- 下面是一個因沒有撰寫拷貝建構元而發生錯誤的例子

```
01 // prog16_9, 錯誤的範例，子類別裡沒有撰寫拷貝建構元
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 class CWin // 定義 CWin 類別，在此為父類別
06 {
07     protected:
08         char id;
09
10     public:
11         CWin(char i='D'):id(i) // 父類別的建構元
12         {
13             cout << "CWin()建構元被呼叫了..." << endl;
14         }
```



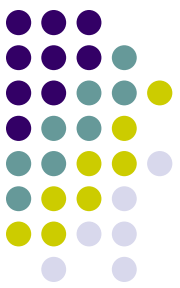
錯誤的使用拷貝建構元 (2/6)

```
15      CWin(const CWin& win)      // 父類別的拷貝建構元
16      {
17          cout << "CWin()拷貝建構元被呼叫了..." << endl;
18          id=win.id;
19      }
20      ~CWin()                    // 父類別的解構元
21      {
22          cout << "CWin()解構元被呼叫了... " << endl;
23          system("pause");
24      }
25 };
26
27 class CTextWin : public CWin // 定義 CTextWin 類別，繼承自 CWin 類別
28 {
29     private:                    // 子類別裡的私有成員
30         char *text;
31
32     public:
33         CTextWin(char i,char *tx):CWin(i)      // 子類別的建構元
34         {
35             cout << "CTextWin()建構元被呼叫了..." << endl;
36             text= new char[strlen(tx)+1];
37             strcpy(text,tx);
38         }
```



錯誤的使用拷貝建構元 (3/6)

```
39     ~CTextWin()
40     {
41         delete [] text;                // 釋放 text 指標指向的記憶體
42         cout << "CTextWin()解構元被呼叫了... " << endl;
43         system("pause");
44     }
45     void show member()                  // 子類別的 show member() 函數
46     {
47         cout << "Window " << id << ": ";
48         cout << "text = " << text << endl;
49     }
50     void set member(char i, char *tx) // 子類別的 set member() 函數
51     {
52         id=i;
53         delete [] text;                // 將原先指向的記憶體釋放
54         text= new char[strlen(tx)+1];  // 重新配置記憶體
55         strcpy(text,tx);
56     }
57 };
58 int main(void)
59 {
60     CTextWin tx1('A',"Hello C++");    // 建立子類別的物件 tx1
61     CTextWin tx2(tx1);                 // 以 tx1 建立子類別的物件 tx2
62 }
```



錯誤的使用拷貝建構元 (4/6)

```

63     tx1.show member();
64     tx2.show member();
65
66     cout << "更改 tx1 物件的成員之後..." << endl;
67     tx1.set member('B', "Welcome C++");
68
69     tx1.show member();
70     tx2.show member();
71
72     system("pause");
73     return 0;
74 }
```

/* prog16_9 OUTPUT-----

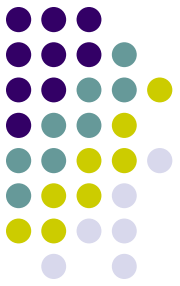
```

CWin() 建構元被呼叫了...
CTextWin() 建構元被呼叫了...
CWin() 拷貝建構元被呼叫了...
Window A: text = Hello C++
Window A: text = Hello C++
更改 tx1 物件的成員之後...
Window B: text = Welcome C++
Window A: text = Welcome C++
請按任意鍵繼續 . . .
CTextWin() 解構元被呼叫了...
請按任意鍵繼續 . . .
CWin() 解構元被呼叫了...
請按任意鍵繼續 . . .
CTextWin() 解構元被呼叫了...
請按任意鍵繼續 . . .
CWin() 解構元被呼叫了...
請按任意鍵繼續 . . .
```

} 執行完 60 行之後的結果

—— 執行完 61 行之後的結果

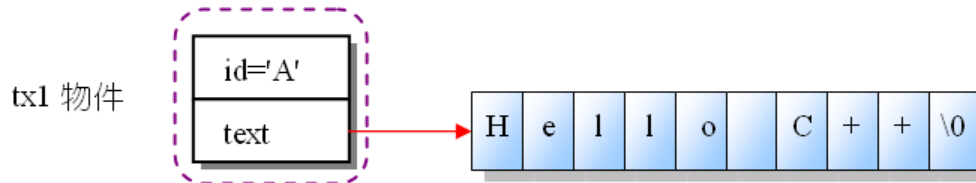
-----*/



錯誤的使用拷貝建構元 (5/6)

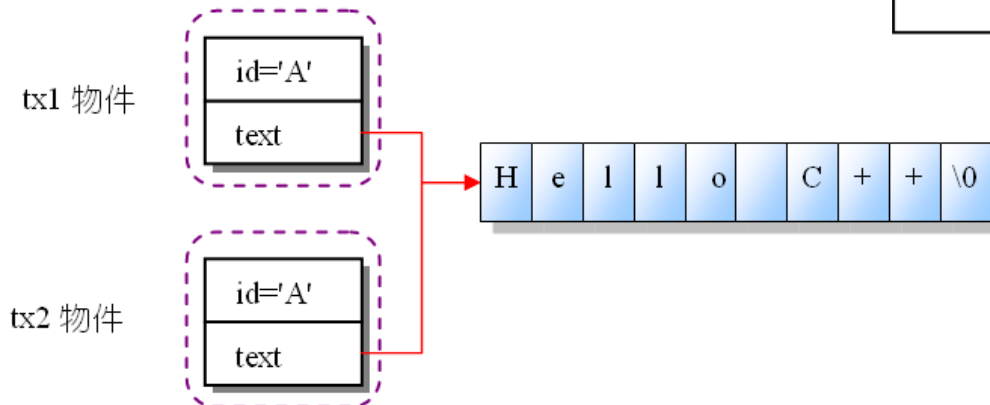
- 下圖為執行完60行後的結果

執行完 60 行之後的結果



- 下圖為執行完61行後的結果

執行完 61 行之後的結果



```
58 int main(void)
59 {
60     CTextWin tx1('A', "Hello C++");
61     CTextWin tx2(tx1);
62     ...
}
```

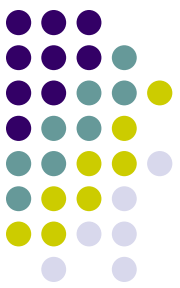



錯誤的使用拷貝建構元 (6/6)

- 要解上面的錯誤，只要在子類別內加上一個拷貝建構元

呼叫父類別的拷貝建構元 傳入 tx 物件

```
CTextWin(const CTextWin &tx) : CWin( tx )    // 子類別的拷貝建構元
{
    cout<<"CTextWin() 拷貝建構元被呼叫了..."<<endl;
    text= new char[strlen(tx.text)+1];  // 另外配置記憶空間給新建的物件
    strcpy(text,tx.text);
}
```



修正錯誤 (1/5)

- 下面的程式是修正沒有撰寫拷貝建構元的錯誤

```
01 // prog16_10, 修正 prog16_9 沒有撰寫拷貝建構元的錯誤
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05
06 // 將 prog16_9 的 CWin 類別定義放在這兒
07 class CTextWin : public CWin // 定義 CTextWin 類別，繼承自 CWin 類別
08 {
09     private:                                // 子類別裡的私有成員
10         char *text;
11
12     public:
13         CTextWin(char i, char *tx):CWin(i)    // 子類別的建構元
14         {
15             cout << "CTextWin() 建構元被呼叫了..." << endl;
16             text= new char[strlen(tx)+1];
17             strcpy(text, tx);
18         }
19         CTextWin(const CTextWin &tx):CWin(tx) // 子類別的拷貝建構元
20         {
21             cout << "CTextWin() 拷貝建構元被呼叫了..." << endl;
22             text= new char[strlen(tx.text)+1];
23             strcpy(text, tx.text);
24         }
```



修正錯誤 (2/5)

```
25     ~CTextWin()
26     {
27         delete [] text;           // 釋放 text 指標所指向的記憶體
28         cout << "CTextWin() 解構元被呼叫了... " << endl;
29         system("pause");
30     }
31     void show member()             // 子類別的 show member() 函數
32     {
33         cout << "Window " << id << ": ";
34         cout << "text = " << text << endl;
35     }
36     void set member(char i, char *tx) // 子類別的 set member() 函數
37     {
38         id=i;
39         delete [] text;           // 將原先指向的記憶體釋放
40         text= new char[strlen(tx)+1]; // 重新配置記憶體
41         strcpy(text,tx);
42     }
43 };
44 int main(void)
45 {
46     CTextWin tx1('A',"Hello C++"); // 建立子類別的物件
47     CTextWin tx2(tx1);
48 }
```



修正錯誤 (3/5)

```
49     tx1.show member();  
50     tx2.show member();  
51  
52     cout << "更改 tx1 物件的成員之後..." << endl;  
53     tx1.set member('B', "Welcome C++");  
54  
55     tx1.show member();  
56     tx2.show member();  
57  
58     system("pause");  
59     return 0;  
60 }
```



修正錯誤 (4/5)

/* prog16_10 OUTPUT-----

```
CWin() 建構元被呼叫了...
CTextWin() 建構元被呼叫了...
CWin() 拷貝建構元被呼叫了...
CTextWin() 拷貝建構元被呼叫了...
Window A: text = Hello C++
Window A: text = Hello C++
更改 tx1 物件的成員之後...
Window B: text = Welcome C++
Window A: text = Hello C++
請按任意鍵繼續 . . .
CTextWin() 解構元被呼叫了...
請按任意鍵繼續 . . .
CWin() 解構元被呼叫了...
請按任意鍵繼續 . . .
CTextWin() 解構元被呼叫了...
請按任意鍵繼續 . . .
CWin() 解構元被呼叫了...
請按任意鍵繼續 . . .
```

} 執行 46 行之後的結果

} 執行 47 行之後的結果

```
44  int main(void)
45  {
46      CTextWin tx1('A', "Hello C++");
47      CTextWin tx2(tx1);
48
49      tx1.show_member();
50      tx2.show_member();
51
52      cout << "更改 tx1 物件的成員之後...";
53      tx1.set_member('B', "Welcome C++");
    ...
}
```

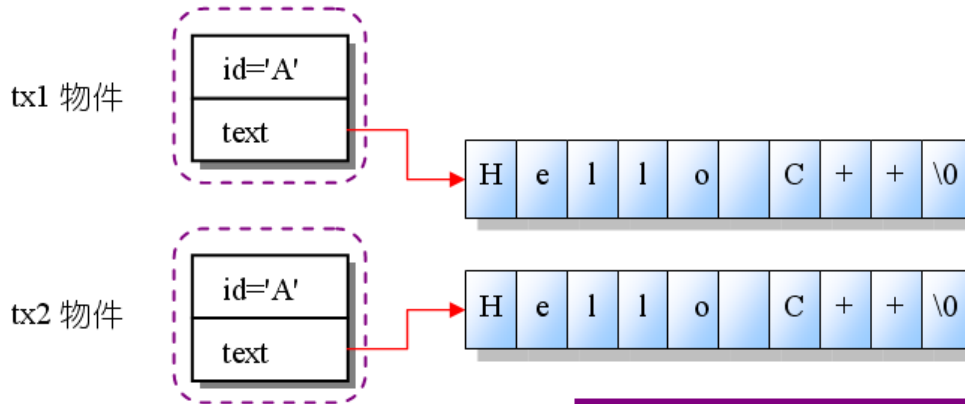
-----*/



修正錯誤 (5/5)

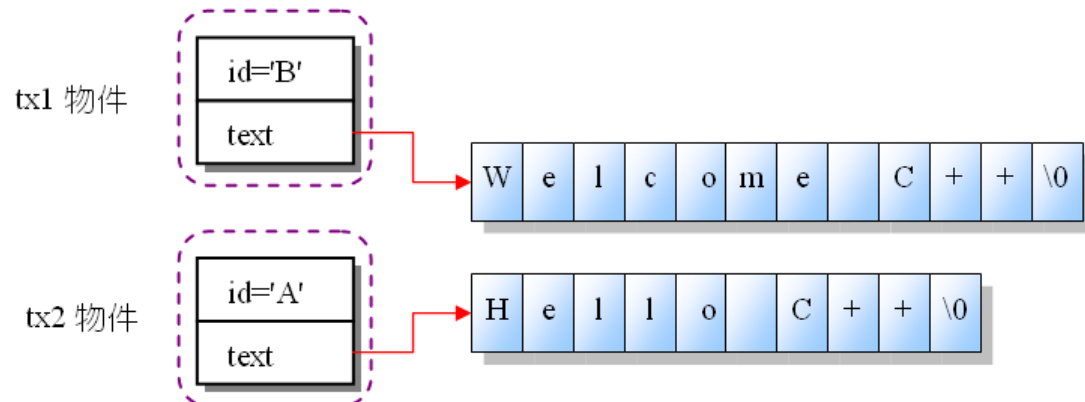
- 下圖為執行47與53行後之結果

執行完 47 行之後的結果



```
44 int main(void)
45 {
46     CTextWin tx1('A', "Hello C++");
47     CTextWin tx2(tx1);
    ...
52     cout << "更改 tx1 物件的成員之後...";
53     tx1.set_member('B', "Welcome C++");
    ...
}
```

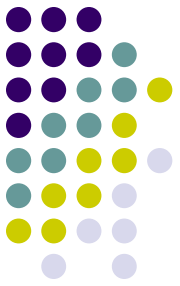
執行完 53 行之後的結果





建構元與解構元的呼叫時機

- 建立物件時，父類別的建構元會先被執行，然後再執行子類別的建構元
- 銷毀物件時，子類別的解構元會先被執行，然後再執行父類別的解構元



-The End-