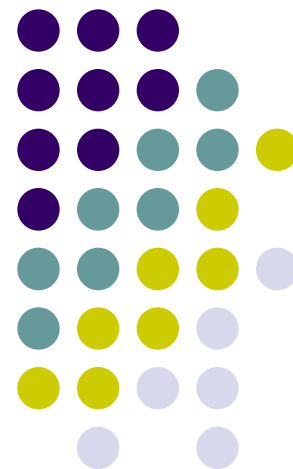


第十九章

例外處理與樣板

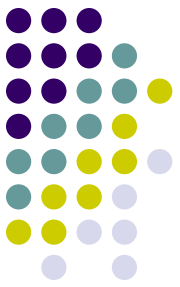
認識例外的基本概念
學習例外發生時的處理
熟悉樣板的操作





例外的基本觀念 (1/2)

- 在撰寫程式時，常見的不尋常狀況如下：
 - 要開啟的檔案並不存在
 - 除數為零。
 - 在存取陣列時，陣列的註標值超過陣列容許的範圍
 - 原本預期使用者由鍵盤輸入的是整數，但使用者輸入的卻是英文字母
 - 系統資源耗盡或是儲存資料的磁碟空間不足，造成程式無法繼續儲存資料
- 這些不尋常的狀況稱為例外（exception）



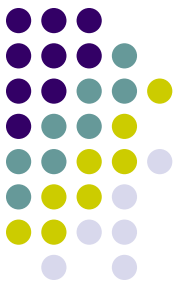
例外的基本觀念 (2/2)

- 沒有撰寫例外的程式碼時，預設的處理機制可能會有下列幾種方式：
 - 直接結束程式
 - 當機
 - 發出警告訊息，然後正常結束執行
 - 自行跳過發生錯誤的地方，繼續執行程式，但是後面的執行可能沒有意義
 - 告訴使用者例外發生的情況
- 在沒有例外處理的語言中，是使用if-else或switch等敘述，來捕捉（catch）程式裡所有可能發生的錯誤



例外處理的程序

- 例外處理是由 `try`與`catch`關鍵字所組成的程式區塊
- `try`區塊內可以撰寫要檢查的程式碼
- 例外發生時，程式的執行便中斷，並由`throw`關鍵字拋出物件給`catch`區塊接收
- 如果在`try`區塊內加上捕捉例外的程式碼，則可針對不同的例外做妥善的處理，這種處理捕捉錯誤的方式稱為例外處理（`exception handling`）



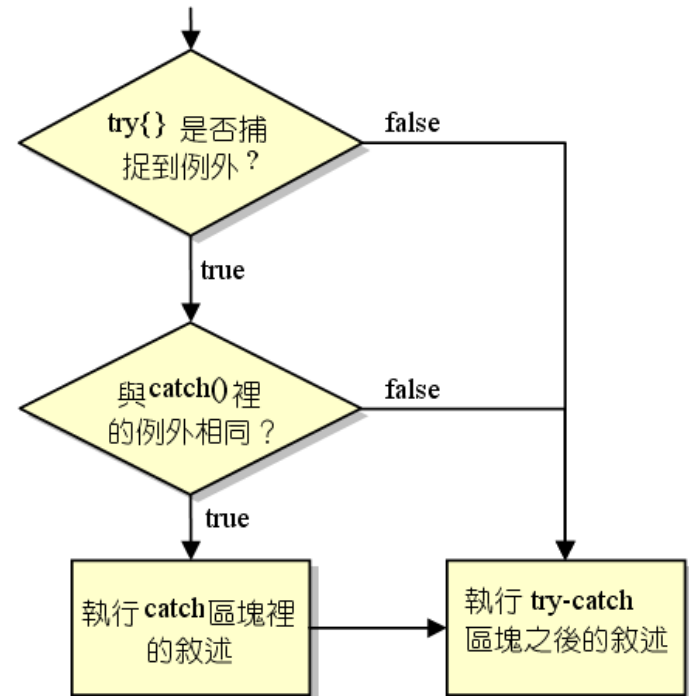
try-catch區塊的語法

- try與catch程式區塊的語法如下

```
try{  
    ...  
    if (要檢查的程式敘述為真)  
        throw 例外物件  
    ...  
}  
catch(型態 變數名稱){  
    例外發生時的處理敘述;  
    ...  
}
```

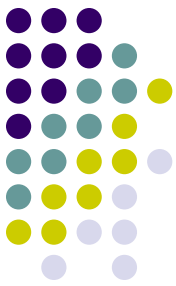
} try 區塊

} catch 區塊



簡單的例外範例

19.1 例外處理



```
01 //prog19_1, 例外的簡單範例, try-catch 區塊的使用
```

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

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

```
04 using namespace std;
```

```
05 int main(void)
```

```
06 {
```

```
07     int array[5];
```

```
08
```

```
09     try                                     // try 區塊，在此區塊內可進行例外的檢查
```

```
10     {
```

```
11         for(int i=0;i<=10;i++)
```

```
12         {
```

```
13             if(i>=5)                        // 若註標值大於等於 5
```

```
14                 throw "Index out of bound"; // 拋出例外
```

```
15             else
```

```
16             {
```

```
17                 array[i]=i*i;
```

```
18                 cout << "array[" << i << "]= " << array[i] << endl;
```

```
19             }
```

```
20         }
```

```
21     }
```

```
22     catch(const char *str)                // catch 區塊
```

```
23     {
```

```
24         cout << "捕捉到" << str << "例外..." << endl;
```

```
25     }
```

```
26
```

```
27     system("pause");
```

```
28     return 0;
```

```
29 }
```

```
/* prog19_1 OUTPUT-----
```

```
array[0]=0
```

```
array[1]=1
```

```
array[2]=4
```

```
array[3]=9
```

```
array[4]=16
```

```
捕捉到 Index out of bound 例外...
```

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

catch 區塊的多載

19.1 例外處理



- catch() 可以多載，以捕捉所有的例外

```
01 //prog19_2, catch 區塊的多載
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 int main(void)
06 {
07     int array[10];
08
09     try
10     {
11         for(int i=0;i<=10;i++)
12         {
13             if(i>9) throw "Index out of bound"; // 拋出字串型態的例外
14             if(i*i>60)
15                 throw i; // 拋出整數型態的例外
16             else
17                 array[i]=i*i;
18         }
19     }
20     catch(const char *str) // 可捕捉字串型態的例外
21     {
22         cout << "捕捉到" << str << "例外..." << endl;
23     }
24     catch(int i) // 可捕捉整數型態的例外
25     {
26         cout << i << "的平方值超過 60 了" << endl;
27     }
28
29     system("pause");
30     return 0;
31 }
```

/* prog19_2 OUTPUT----
8 的平方值超過 60 了
-----*/

捕捉任何型態的例外

19.1 例外處理



- 下面是可以捕捉任何型態之例外的範例

```
01 //prog19_3, 捕捉任何型態的例外
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 int main(void)
06 {
07     int array[10];
08
09     try
10     {
11         for(int i=0;i<=10;i++)
12         {
13             if(i>9)
14                 throw "Index out of range";
15             if(i*i>60)
16                 throw i;
17             else
18                 array[i]=i*i;
19         }
20     }
21     catch(...) // 可接收任何型態的例外
22     {
23         cout << "捕捉到例外了..." << endl;
24     }
25
26     system("pause");
27     return 0;
28 }
```

```
/* prog19_3 OUTPUT---
捕捉到例外了...
-----*/
```




樣板

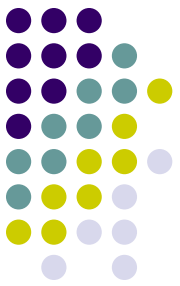
- 「樣板」 (template) 的作用有點類似函數的多載
- C++提供
 - 「函數樣板」 (function template)
 - 「類別樣板」 (class template)



多載的複習

- 下面的範例定義一個 `add()` 函數的多載

```
01 // prog19_4, 多載的複習
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 int add(int a, int b)
06 {
07     return a+b;
08 }
09
10 double add(double a, double b)
11 {
12     return a+b;
13 }
14
15 int main(void)
16 {
17     cout << "add(3,4)=" << add(3,4) << endl;
18     cout << "add(3.2,4.6)=" << add(3.2,4.6) << endl;
19
20     system("pause");
21     return 0;
22 }
```



函數樣板

- 「函數樣板」是指將具有相同程式碼的函數撰寫成一個樣板，而把其中引數不同型態之處，用「型態變數」來取代
- 以「函數樣板」製作新函數的過程，稱為「樣板的具體化」（instantiation of template）
- 下面列出定義「函數樣板」的語法

```
template <class 型態變數 1, class 型態變數 2,...>  
傳回型態 函數名稱 (型態變數 引數 1, 型態變數 引數 2,...)  
{  
    函數主體  
}
```



函數樣板的使用範例 (1/6)

- 下面是add 的函數樣板

```
01  template <class T>           // 定義函數樣板，其中填入樣板的型態只有一種
02  T add(T a,T b)              // add() 的傳回型態為 T，傳入的引數型態都是 T
03  {
04      T sum=a+b;              // 設定變數 sum 的型態為 T，其值等於 a+b
05      return sum;             // 傳回 sum 的值
06  }
```

- 我們也可以把add函數樣板變成如下的寫法

```
01  template <class T> T add(T a,T b)    // 定義函數樣板 add()
02  {
03      T sum=a+b;
04      return sum;
05  }
```



函數樣板的使用範例 (2/6)

- 如果想計算整數的加法，則可以用下面的語法

```
add <int> (3, 4);    // 將整數 3 和 4 傳入 add() 函數
```

└ 此處設定函數樣板中，第一行的
型態變數 T 均會以 `int` 來取代

- 當程式執行到 `add<int>(3,4)` 時，便會把 `add()` 函數視為下面的定義來呼叫它

```
int add(int a, int b) // 將所有的型態變數 T 均置換成 int
{
    int sum=a+b;
    return sum;
}
```

- 想利用 `add()` 函數把兩個倍精度浮點數相加

```
add <double> (3.2, 4.6);
```

└ 此處設定函數樣板中，第一行的
型態變數 T 均會以 `double` 來取代



函數樣板的使用範例 (3/6)

- 下面是使用函數樣板之完整範例

```
01 //prog19_5, 函數樣板的使用範例
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 template <class T> // 定義函數樣板
06 T add(T a,T b) // add() 的傳回型態為 T，傳入的兩個引數型態也是 T
07 {
08     T sum=a+b; // 設定變數 sum 的型態為 T，其值等於 a+b
09     return sum;
10 }
11
12 int main(void)
13 {
14     cout << "add(3,4)=" << add<int>(3,4) << endl;
15     cout << "add(3.2,4.6)=" << add<double>(3.2,4.6) << endl;
16
17     system("pause");
18     return 0;
19 }
```

```
/* prog19_5 OUTPUT----
add(3,4)=7
add(3.2,4.6)=7.8
-----*/
```



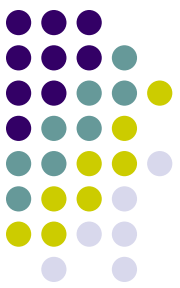
函數樣板的使用範例 (4/6)

- 下面的範例省略add函數之後<>括號

```
01 //prog19_6, 括號的省略
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 template <class T>                                // 定義函數樣板 add()
06 T add(T a,T b)
07 {
08     T sum=a+b;
09     return sum;
10 }
11
12 int main(void)
13 {
14     cout << "add(3,4)=" << add(3,4) << endl; // 呼叫 add(3,4)
15     cout << "add(3.2,4.6)=" << add(3.2,4.6) << endl; // 呼叫 add(3.2,4.6)
16
17     system("pause");
18     return 0;
19 }
```

/* prog19_6 OUTPUT---

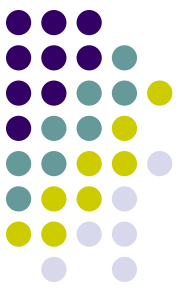
add(3,4)=7
add(3.2,4.6)=7.8
-----*/



函數樣板的使用範例 (5/6)

- 函數樣板也允許程式設計者填入多個型態不同的引數

```
01 //prog19_7, 樣板的使用範例(引數型態不同時)
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 template <class T1, class T2>                // 定義函數樣板
06 double average(T1 a,T2 b) // 定義 average(), 可接收 T1 與 T2 型態的變數
07 {
08     cout << "sizeof(a)= " << sizeof(a) << ", ";
09     cout << "sizeof(b)= " << sizeof(b) << endl;
10     return (double)(a+b)/2;                // 傳回變數 a,b 的平均值
11 }
12
13 int main(void)
14 {
15     cout << "average(3,4.2)= " << average<int,double>(3,4.2) << endl;
16     cout << "average(5.7,12)= " << average<double,int>(5.7,12) << endl;
17
18     system("pause");
19     return 0;
20 }
```

函數樣板的使用範例 (6/6)

```
/* prog19_7 OUTPUT-----
sizeof(a)= 4, sizeof(b)= 8
average(3,4.2)= 3.6
sizeof(a)= 8, sizeof(b)= 4
average(5.7,12)= 8.85
-----*/
```

```
template <class T1, class T2 >
double average(T1 a, T2 b) {
    ...
}

int main(void)
{
    ...
    average<int, double>(3,4.2);
    ...
}
```

- add函數之後<>括號也可以省略

```
15 cout << "average(3,4.2)= " << average(3,4.2) << endl;
16 cout << "average(5.7,12)= " << average(5.7,12) << endl;
```



類別樣板的定義 (1/2)

- 類別樣板定義的格式如下

```
template <class 型態變數 1, class 型態變數 2, ...>  
class 類別名稱  
{  
    資料成員與成員函數的定義  
};
```

- 利用類別樣板來建立物件時，可利用如下的語法

```
類別名稱 <型態 1, 型態 2, ...> 物件名稱;
```



類別樣板的定義 (2/2)

- 下面的程式碼定義CWin的類別樣板

```
01  template <class T>                // 定義CWin的類別樣板
02  class CWin
03  {
04      protected:
05          T width, height;          // 宣告資料成員
06
07      public:
08          CWin(T w,T h):width(w),height(h){};    // 建構元
09
10          void show(void);           // show()的原型
11  };
```

- 下面的兩行敘述分別建立資料成員

```
CWin <int> win1(50,60);           // 建立win1物件，並設定資料成員為 int
CWin <double> win2(50.25,60.74);  // 設定資料成員的型態為 double
```



類別樣板之外的函數定義

- 在類別樣板之外的函數定義格式

```
template <class 型態變數 1, class 型態變數 2, ...>  
傳回型態 類別名稱<型態變數 1, 型態變數 2, ...>::函數名稱(引數...)  
{  
    函數的定義  
};
```

- 依據上面的格式，可把show() 函數撰寫如下

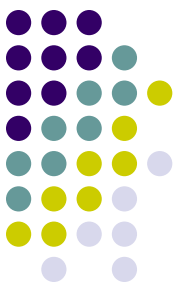
```
01  template <class T>  
02  void CWin<T>::show()  
03  {  
04      cout << "width=" << width << ", ";  
05      cout << "height=" << height << endl;  
06  }
```



類別樣板的使用範例1 (1/2)

- prog19_8是類別樣板的使用範例

```
01 //prog19_8, 類別樣板的使用範例
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 template <class T> // 定義類別樣板
06 class CWin
07 {
08     protected:
09         T width, height; // 宣告資料成員
10
11     public:
12         CWin(T w,T h):width(w),height(h){}; // 建構元
13
14         void show(void); // show() 函數的原型
15 };
16
17 template <class T> // 定義 show() 函數
18 void CWin<T>::show()
19 {
20     cout << "width=" << width << ", ";
21     cout << "height=" << height << endl;
22 }
```



類別樣板的使用範例1 (2/2)

```
23
24  int main(void)
25  {
26      CWin <int> win1(50,60);           // 建立 win1 物件
27      CWin <double> win2(50.25,60.74); // 建立 win2 物件
28
29      cout << "win1 object: ";
30      win1.show();
31      cout << "win2 object: ";
32      win2.show();
33
34      system("pause");
35      return 0;
36  }
```

```
/* prog19_8 OUTPUT-----
win1 object: width=50, height=60
win2 object: width=50.25, height=60.74
-----*/
```



類別樣板的使用範例2 (1/2)

- 下面的範例是使用兩個型態變數的例子

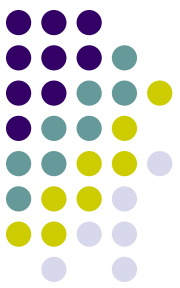
```
01 //prog19_9, 類別樣板的使用範例 (不同引數型態的情形)
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 template <class T1,class T2>           // 定義類別樣板
06 class CWin
07 {
08     protected:
09         T1 width;                     // 指定 width 的型態為 T1
10         T2 height;                   // 指定 height 的型態為 T2
11     public:
12         CWin(T1 w,T2 h):width(w),height(h){};
13
14         void show(void);
15 };
16
```



類別樣板的使用範例2 (1/2)

```
17  template <class T1, class T2>
18  void CWin<T1,T2>::show()           // 定義 show() 函數
19  {
20      cout << "width=" << width << ", ";
21      cout << "height=" << height << endl;
22  }
23
24  int main(void)
25  {
26      CWin <int,double> win1(50,60.05);    // 建立 win1 物件
27      CWin <double,int> win2(50.25,74);    // 建立 win2 物件
28
29      cout << "win1 object: ";
30      win1.show();
31      cout << "win2 object: ";
32      win2.show();
33
34      system("pause");
35      return 0;
36  }
```

/* prog19_9 OUTPUT-----
win1 object: width=50, height=60.05
win2 object: width=50.25, height=74
-----*/



類別樣板的使用範例3 (1/2)

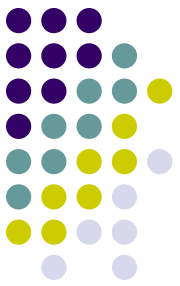
- 下面的範例限定樣板內某個引數必須為某個型態的變數

```
01 //prog19_10, 樣板內限定某個引數必須為某個型態的變數
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 template <class T1,class T2,char id='D'> // 樣板內加入特定型態的變數
06 class CWin
07 {
08     protected:
09         T1 width;
10         T2 height;
11     public:
12         CWin(T1 w,T2 h):width(w),height(h){};
13
14         void show();
15 };
16
```

/* prog19_10 OUTPUT-----

```
win1 object: A
width=50, height=60.05
win2 object: D
width=50.25, height=74
```

-----*/



類別樣板的使用範例3 (1/2)

```
17  template <class T1, class T2, char id>          // 定義 show() 函數
18  void CWin<T1,T2,id>::show()
19  {
20      cout << id << endl;
21      cout << "width=" << width << ", ";
22      cout << "height=" << height << endl;
23  }
24
25  int main(void)
26  {
27      CWin <int,double,'A'> win1(50,60.05);
28      CWin <double,int> win2(50.25,74);
29
30      cout << "win1 object: ";
31      win1.show();
32      cout << "win2 object: ";
33      win2.show();
34
35      system("pause");
36      return 0;
37  }
```

注意此處必須填上變數名稱，而非資料型態

```
/* prog19_10 OUTPUT-----
win1 object: A
width=50, height=60.05
win2 object: D
width=50.25, height=74
-----*/
```



樣板的特殊化 (1/2)

- 樣板的特殊化可分為
 - 「類別樣板的特殊化」
 - 「函數樣板的特殊化」
- 以CWin類別為例，可將它定義成類別樣板

```
template <class T>                                // 類別樣板
class CWin
{
    T width, height;                               // 資料成員
public:
    CWin(T w,T h):width(w),height(h){};           // 建構元
    T area(void){ return width*height; }          // 成員函數 area()
};
```



樣板的特殊化 (2/2)

- 將CWin類別樣板裡的函數特殊化，成為特殊化的類別樣板

定義特殊化的類別樣板，必須
在樣板前加上 `template <>`

欲特殊化的型態

```
template <> class CWin <int>           // 特殊化的類別樣板
{
    int width, height;                 // 資料成員
public:
    CWin(int w, int h):width(w),height(h){}; // 建構元
    int area(void){ return 0; }           // 成員函數
};
```

資料成員的型態須和欲特殊化的型態相同



樣板的特殊化之範例1 (1/2)

- prog19_11是類別樣板特殊化的範例

```
01 //prog19_11, 類別樣板的特殊化
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
```

```
05 template <class T>                                // 類別樣板
06 class CWin
07 {
08     T width,height;
09     public:
10         CWin(T w,T h):width(w),height(h){};
11
12         T area(void){ return width*height; }
13 };
```

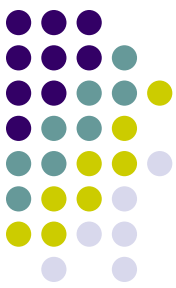
```
14
15 template <> class CWin <int>                        // 特殊化的類別樣板
16 {
17     int width,height;
18     public:
19         CWin(int w, int h):width(w),height(h){};
20
21         int area(void){ return 0; }
22 };
```



樣板的特殊化之範例1 (2/2)

```
23
24  int main(void)
25  {
26      CWin <int> win1(50,60);
27      CWin <double> win2(12.3,45.8);
28      CWin <short> win3(12,45);
29
30      cout << "win1 object: ";
31      cout << win1.area() << endl;
32
33      cout << "win2 object: ";
34      cout << win2.area() << endl;
35
36      cout << "win3 object: ";
37      cout << win3.area() << endl;
38
39      system("pause");
40      return 0;
41  }
```

```
/* prog19_11 OUTPUT-----
win1 object: 0
win2 object: 563.34
win3 object: 540
-----*/
```



樣板的特殊化之範例2 (1/2)

```
01 //prog19_12, 類別樣板之成員函數的特殊化
02 #include <iostream>
03 #include <cstdlib>
04 using namespace std;
05 template <class T>
06 class CWin
07 {
08     T width,height;
09     public:
10         CWin(T w,T h):width(w),height(h){};
11
12         T area(void)
13         {
14             return width*height;
15         }
16 };
17
```

```
18 template <> int CWin<int>::area(void) // 類別樣板內成員函數的特殊化
19 {
20     return 0;
21 }
```

- 類別樣板內成員函數的特殊化的範例

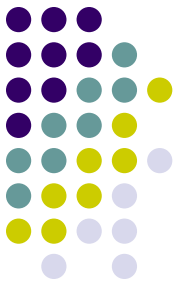
// 類別樣板



樣板的特殊化之範例2 (2/2)

```
22
23  int main(void)
24  {
25      CWin <int> win1(50,60);
26      CWin <double> win2(12.3,45.8);
27
28      cout << "win1 object: ";
29      cout << win1.area() << endl;
30
31      cout << "win2 object: ";
32      cout << win2.area() << endl;
33
34      system("pause");
35      return 0;
36  }
```

```
/* prog19_12 OUTPUT-----
win1 object: 0
win2 object: 563.34
-----*/
```

The End-