Chapter 16

異常處理

程式異常

- 語法錯誤
- 執行錯誤
  - 除法的分母為零
  - 指標跑出範圍，設定數值
  - 數值溢位
  - . . .

} 異常

syntax error, run-time error, exception

傳統異常處理方式

- 範例

```
double a , b ;
cin >> a >> b ;
cout << ”比值：” << a / b << endl ;

double a , b ;
cin >> a >> b ;
if( b == 0 )
   cout << ”除法的分母不得為零” << endl ;
else
   cout << ”比值 : ” << a / b << endl ;
```

```
#include <assert.h>                    // 或者為 <cassert>
...
cin >> a >> b ;
                                       // 確認分母 b 不能為零
assert( b != 0 ) ;
cout << ”比值 : ” << a / b << endl ;
```

C++異常處理機制

- 異常處理機制包含
  - 測試區：
    由 try 區塊所圍成的測試區間
  - 異常處理區：
    由 catch 所設定的異常處理程式區間

異常處理程序

- 當程式進入 **try** 測試區內執行時，若程式偵測到將發生異常，可以擲出 (throw) 異常處理物件後，程式隨即轉往對應的異常處理區處理

```
Err FOO ;
try {
    ...
}
```

throw FOO →

```
catch( Err e ) {
    ...
}
```

❖ 以上的 **Err** 為使用者自定的異常處理類別

異常處理程式範例


```
class ERR {
   public :
      char* err_mesg() const { return "下標超出範圍" ; }
};

int main(){
   const int M = 5 ;
   int number[M] , i , no , count = 1 ;
   while ( count <= M ){
      try {
         cout << "輸入第" << count << "資料 : " ;
         cin >> i >> no ;
         if( i < 0 || i >= M ) throw ERR() ;   // 如果下標不在範圍內，則擲出 ERR 物件
         number[i] = no ;
         count++ ;
      }
      catch( ERR e ) {
         cout << e.err_mesg() << endl ;
      }
   }
   for( i = 0 ; i < M ; ++i ) cout << i << ' ' << number[i] << endl ;
   ...
}
```

異常處理類別設計（一）

- 使用者可以根據須要自行設計異常處理類別

```
class ERR {
   private :
      int kind ;
   public :
      ERR( int i ) : kind(i) {}
      char* err_mesg() const {
          if( kind == 1 ) return ”下標超出範圍” ;
          if( kind == 2 ) return ”陣列元素所儲存數值不是偶數” ;
          ...
      }
} ;
```

❖ 對複雜的問題而言，甚至可以設計出一異常處理類別架構

異常處理類別設計（二）

- 程式可以根據異常的種類擲出不同的異常處理物件

```
While ( count <= M ) {
   try {
      cout << "輸入第 " << count << " 資料 : " ;
      cin >> i >> no ;

      // 決定造成異常的種類
      if( i < 0 || i >= M ) throw ERR(1) ;
      if( no % 2 == 1 ) throw ERR(2) ;

      number[i] = no ;
      count++ ;
   }
   catch( ERR e ) cout << e.err_mesg() << endl ;
}
```

❖ 在此 **try** 區塊內的兩個擲出 (throw) 敘述都擲出同樣的 **ERR** 類別物件，因此只須使用一個接收 (catch) 區塊即可

異常處理類別設計（三）

- 異常處理類別甚至可以簡化成一空類別，此時接收區就不須註明使用的類別物件名稱

```
class ERR{} ;

unsigned int square( unsigned int no ) {
     if( no > 65535 ) throw ERR() ;
     return no * no ;
}

int main() {
   int no ;
   cout << "> 輸入一個正整數 : " ;
   cin >> no ;

   try { cout << "平方為 : " << square(no) << endl ; }

   catch( ERR ) { cout << no << " : 整數太大 無法處理" ; }
}
```

中途結束函式：exit

- 如果程式須在中途結束，則可使用中途結束函式

```
void exit(int) ;
```

函式可以透過參數告知作業系統異常中斷的整數代碼，一般而言， 0 代表正常結束

```
int i ;
cin >> i ;
if( i == 0 ){
     cout << ” i 不得為 0 ” ;
     exit(1) ;
} else {
    ...
}
```

一般函式的異常處理（一）

- 若函式內有擲出異常處理物件，可在函式參數列後加註 **throw** 字樣

```
struct ERR {
   char* err_mesg() { return "數字太大" ; }
} ;

unsigned square( unsigned no ) throw(ERR) {
   if( no > 65535 ) throw ERR() ;
   return no * no ;
}

int main(){
   unsigned int i = 300000 ;
   try {
      cout << square(i) << endl ;
   }
   catch( ERR e ) { cout << e.err_mesg() << endl ; }
}
```

一般函式的異常處理（二）

- 若函式中可能擲出不同類別的異常處理物件，則可在名稱之間以逗號分開，都放在 throw 的小括號內

```
struct Err1 { . . . } ;
struct Err2 { . . . } ;

int fn( int a ) throw(Err1,Err2) {
     if( a == 1 ) throw(Err1) ;
     if( a == 2 ) throw(Err2) ;
     ...
}
```

- 若函式內並無任何擲出敘述則可以 throw() 標明

```
unsigned remainder( unsigned a ) throw() {
   return ( a % 10 ) ;
}
```

類別內成員函式的異常處理

- 類別成員函式的 throw 標幟擺放位置與普通函式相同，但若為建構函式，則擲出的標幟需在初值設定列之前

```
class Err1 { ... } ;
class Err2 { ... } ;

class FOO {
   private :
      int no ;
      ...
   public :

      FOO(int i) throw(Err1) : no(i) {...}  // 包含一個異常處理物件 Err1

      // 包含兩個異常處理物件 Err1,Err2
      int square() const throw(Err1,Err2) {...}

      void print() const throw() {...}      // 沒有使用異常處理機制
      ...
} ;
```

多個擲出與多個接收

- 在 try 區塊內也有可能擲出不同類型的異常處理物件，因此也須要多個接收區塊

```
int i ;
while(1){
   try {
      Fraction foo , bar ;
      cin >> i ;
      if( i > 100 ) throw ERR1() ;
      if( i < 0 )  throw ERR2() ;
      敘述 1 ;
   }
   catch( ERR1 e ) { 敘述 2 ; }
   catch( ERR2 e ) { 敘述 3 ; }
   敘述 4 ;
}
```

- 當程式正常執行完某一接收 (catch) 區塊時，程式隨即跳往最後一個接收區塊後繼續執行，執行並不會因此而中斷

統收異常處理機制（一）

- 統收異常處理機制：

  可用來接收所有型別的擲出物件

```
catch ( ... ) {
    敘述 ;
}
```

❖ 統收異常處理機制利用三個句點來表示接收區塊可處理所有的擲出物件

catch-all handler

統收異常處理機制（二）

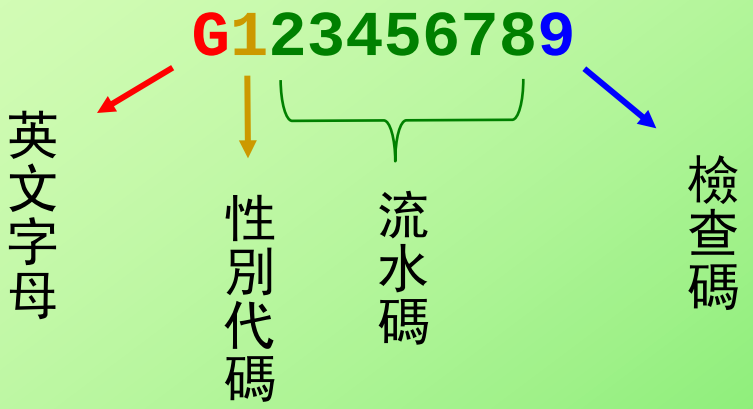
- 統收異常處理機制可與其它接收敘述合用，但通常被擺放在所有接收區塊之後，用以接收未能被其他接收區塊處理的物件，其效果有如備用的異常接收區塊

```
try {
   if( 敘述 1 ) throw ERR1() ;
   if( 敘述 2 ) throw ERR2() ;
}

catch( ERR2 e ) { 敘述 3 ; }
catch(  ...   )  { 敘述 4 ; }
```

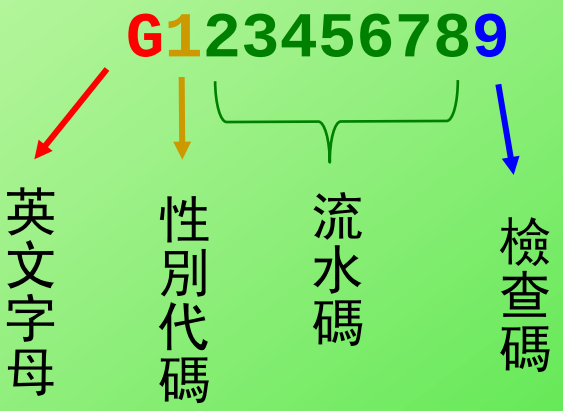
範例：證件號碼驗證（一）

- 證件號碼



- 驗證方式
  - 將英文字母 A-Z 分別以 1,2,...,26 號碼對應
  - 展開流水碼的個別數字後再分別乘上加權數

範例：證件號碼驗證（二）



■ 若預設 **1756179135** 為加權數

G 性別 流水碼

```
    0   7   1   2   3   4   5   6   7   8
x   1   7   5   6   1   7   9   1   3   5
-------------------------------------------
    0 + 49 + 5 + 12 + 3 + 28 + 45 + 6 + 21 + 40
=  209
```

取出其個位數 **9**，此個位數即為檢查碼

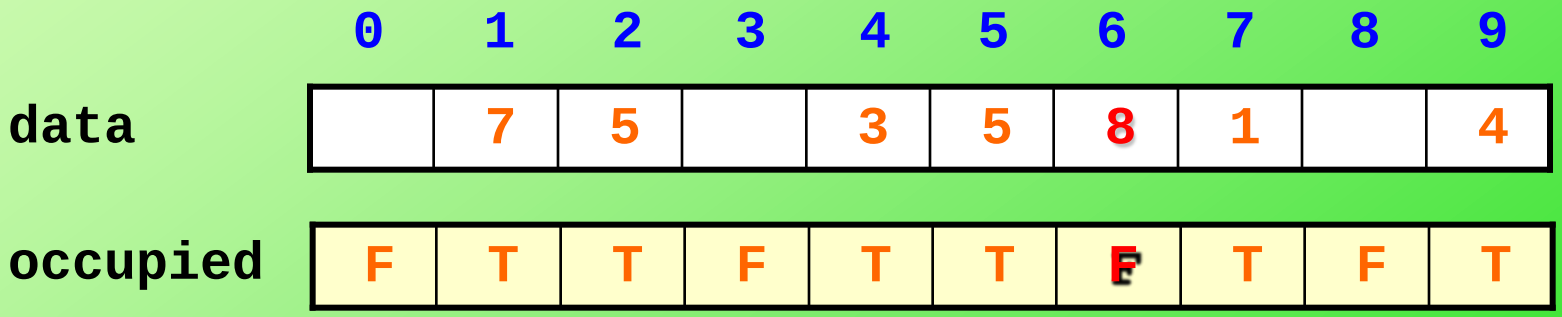
程式 輸出

範例：樣板集合類別

- 集合可用來儲存元素，集合元素並不須相鄰

```
(1) int  count                    : 記錄集合內元素的個數
(2) T    data[Set_Size]           : 用來儲存集合的元素
(3) bool occupied[Set_Size]       : 記錄 data 陣列某元素位置
                                    是否已經儲存元素
```

若集合的 data 陣列可儲存 10 個元素，則



以上 data[6] 所對應 occupied[6] 為 false，所以並非集合的元素

程式　輸出