第13章 例外處理(Exception Handling)

13.1 簡介

- 例外
  - 表示程式出問題
  - 發生不正常情況
  - 程式停止執行
- 例外處理
  - 解決例外情況
  - 程式可以繼續執行-可容許錯誤
    - 例如:可處理程式除0錯誤

13.2 例外處理概觀

- 術語(Terminology)
  - 丟出例外(throws an exception)
    - 函式遇到錯誤
  - 例外處理程式(Exception handler)
    - 捕捉及處理例外
  - 假如沒有例外處理程式捕捉適當例外
    - 程式將可能中斷執行

13.2 例外處理概觀

- C++程式

```
try {
  可能丟出例外的程式碼
  使用throw丟出例外, 可丟出任何資料型態的物件
  通常丟出exception物件
}
catch (exceptionType1){
  處理例外的程式碼
}
catch (exceptionType2){
  ...
}
```

13.4 簡單的例外處理範例:除0錯誤

- 例外類別
  - 基礎類別**exception,**
    - 引用**<exception>**
  - 建構子可接受一個字串
    - 用來描述例外
  - 成員函式what()會傳回該字串
- 衍生類別
  - runtime_error, logic_error
  - bad_alloc, bad_cast, bad_typeid

13.2 例外處理概觀

- 捕捉所有例外類別
  - catch (...)
  - catch (exception &anyException)
- 捕捉特定例外類別
  - catch (MyException &myEx)
  - catch (bad_alloc &memAllocationEx)
  - catch (runtime_error &error)

13.2 例外處理概觀

- 在try區塊內
  - 假如發生例外
    - 程式會跳過try區塊內還沒執行的程式
    - 進入適當的catch區塊
      - 假如沒有例外處理程式可用
        - 函式結束
        - 尋找較上層的catch區塊
  - 如果沒有例外
    - 執行完try區塊
    - 跳過**`catch`**區塊

13.3 其它錯誤處理技術

- 忽略例外 - 程式可能不正常結束
- 終止程式執行
- 設定錯誤指示燈
- 呼叫exit () - **<cstdlib>**
- 跳到外層處理錯誤 - 破壞程式結構
  - **setjump** and **longjump** - **<csetjmp>**
- 專屬的錯誤處理程式
  - 例如: new有專屬錯誤處理程式

13.4 簡單的例外處理範例:除0錯誤

- 範例程式
  - 定義新的例外類別
    - **`DivideByZeroException`**
    - 繼承exception類別
  - 處理除0錯誤

13.4 簡單的例外處理範例:除0錯誤

- 在除法函式中
  - 測試分母
  - 假如分母為零,丟出例外
- 在try區塊內
  - 呼叫除法函式
  - catch DivideByZeroException 例外物件
- 遇到除0錯誤,程式可繼續執行

Outline

fig13_01.cpp
(1 of 3)

```
1  // Fig. 13.1: fig13_01.cpp
2  // 一個簡單的例外處理範例程式
3  // 用來檢查除0例外
4  #include <iostream>
5
6  using std::cout;
7  using std::cin;
8  using std::endl;
9
10 #include <exception>
11
12 using std::exception;
13
14 // 定義一個例外類別:DivideByZeroException
15 // 當發生除0錯誤時可丟出此類例外
16 class DivideByZeroException : public exception {
17
18 public:
19
20    // 建構子:直接實作,指定錯誤訊息
21    DivideByZeroException::DivideByZeroException()
22       : exception( "attempted to divide by zero" ) {}
23
24 };  // end class DivideByZeroException
25
```

Outline

fig13_01.cpp
(2 of 3)

```
26 // 執行除法動作
27 // 如果除數為0,則丟出DivideByZeroException物件
28 double quotient( int numerator, int denominator )
29 {
30    // 假如除數為0丟出 DivideByZeroException 例外並結束此函式
31    if ( denominator == 0 )
32       throw DivideByZeroException(); //丟出例外並結束函式執行
33
34    // 傳回除法運算結果
35    return static_cast< double >( numerator ) / denominator;
36
37 }  // end function quotient
38
39 int main()
40 {
41    int number1;    // 使用者指定的被除數
42    int number2;    // 使用者指定的除數
43    double result;  // 除法運算結果
44
45    cout << "Enter two integers (end-of-file to end): ";
46
```

Outline

fig13_01.cpp
(3 of 3)

```
47     // 要求使用者輸入被除數與除數
48     while ( cin >> number1 >> number2 ) {
49
50        // try區塊內含可能傳回例外的程式碼
51        // 以及當例外發生時不應執行的程式碼
52        try {
53           result = quotient( number1, number2 );
54           cout << "The quotient is: " << result << endl;
55
56        } // end try
57
58        // 例外處理程式:處理除0例外
59        catch ( DivideByZeroException &divideByZeroException )
{
60           cout << "Exception occurred: "
61                << divideByZeroException.what() << endl;
62
63        } // end catch
64
65        cout << "\nEnter two integers (end-of-file to end): ";
66
67     }  // end while
68
69     cout << endl;
70
71     return 0;  // 正常結束
72
73  }  // end main
```

```
Enter two integers (end-of-file to end): 100 7
The quotient is: 14.2857

Enter two integers (end-of-file to end): 100 0
Exception occurred: attempted to divide by zero

Enter two integers (end-of-file to end): ^Z
```

Outline

fig13_01.cpp
output (1 of 1)

13.5 重新丟出例外

- 重新丟出例外
  - 當例外處理程式不能完全處理例外時
    - 處理一部份, 再重新丟出
    - 上層的try區塊與相對應的catch區塊會處理
- 要重新丟出例外
  - 使用指令 "throw;"
    - 不需參數
    - 結束函式執行

Outline

fig13_02.cpp
(1 of 2)

```
1   // Fig. 13.2: fig13_02.cpp
2   // 重新丟出例外範例程式
3   #include <iostream>
4
5   using std::cout;
6   using std::endl;
7
8   #include <exception>
9
10  using std::exception;
11
12  // throw, catch and rethrow exception
13  void throwException()
14  {
15     // 丟出例外並利用catch捉住例外
16     try {
17        cout << "  Function throwException throws an exception\n";
18        throw exception();  // 產生例外
19
20     } // end try
21
22     // 處理例外
23     catch ( exception &caughtException ) {
24        cout << "  Exception handled in function throwException"
25             << "\n  Function throwException rethrows exception";
26
27        throw;  // 重新丟出例外
28
29     } // end catch
```

Outline

```
30
31    cout << "This also should not print\n";
32
33 }  // end function throwException
34
35 int main()
36 {
37    // 丟出例外
38    try {
39       cout << "\nmain invokes function throwException\n";
40       throwException();
41       cout << "This should not print\n";
42
43    } // end try
44
45    // 處理例外
46    catch ( exception &caughtException ) {
47       cout << "\n\nException handled in main\n";
48
49    } // end catch
50
51    cout << "Program control continues after catch in main\n";
52
53    return 0;
54
55 }  // end main
```

fig13_02.cpp
(2 of 2)

Outline

```
main invokes function throwException
  Function throwException throws an exception
  Exception handled in function throwException
  Function throwException rethrows exception

Exception handled in main
Program control continues after catch in main
```

fig13_02.cpp
output (1 of 1)

13.6 函式例外清單

- 列出函式可能丟出的例外
  - 例外丟出清單

```
int someFunction( double value )
throw ( ExceptionA, ExceptionB, ExceptionC )
{
   // function body
}
```

  - 函式只能丟出例外
    - ExceptionA, ExceptionB, ExceptionC
    - 或以上例外的衍生類別
    - 如果丟出其它例外, 會呼叫 unexpected函式

13.6 函式例外清單

- 沒有例外丟出清單
  - 可丟出任何種類例外
- 例外丟出清單沒列任何例外
  - 不能丟出例外

13.7 處理非預期(unexpected)例外

- **unexpected**函式
  - 利用**set_unexpected** 函式
    - **<exception>**
    - 預設值會終止程式執行
  - **set_terminate**
    - 設定處理函式
      - 函式沒有輸入
      - 傳回值為**void**
    - 預設函式:**abort**
      - 如果有設定處理函式,處理函式執行完也會呼叫**abort**

13.8 堆疊返回

- 如果例外被丟出但是沒有被捉住
  - 結束目前函式
    - 清除函式呼叫堆疊
  - 搜尋可處理例外的try/catch
    - 如果沒有找到則再回到上一層
- 假如例外都沒被捉住
  - 結束程式執行

Outline

fig13_03.cpp
(1 of 2)

```
1  // Fig. 13.3: fig13_03.cpp
2  // 堆疊重返說明範例程式.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <stdexcept> // 使用runtime_error例外
9
10 using std::runtime_error;
11
12 // function3丟出runtime_error例外
13 void function3() throw ( runtime_error )
14 {
15    throw runtime_error( "runtime_error in function3" ); // 4
16 }
17
18 // function2呼叫function3
19 void function2() throw ( runtime_error )
20 {
21    function3(); // 3
22 }
23
```

Outline

```
24  // function1呼叫function2
25  void function1() throw ( runtime_error )
26  {
27     function2(); // 2
28  }
29
30  // 主程式
31  int main()
32  {
33     // 呼叫function1
34     try {
35        function1(); // 1
36
37     } // end try
38
39     // 處理runtime_error
40     catch ( runtime_error &error ) // 5
41     {
42       cout << "Exception occurred: " << error.what() << endl;
43
44     } // end catch
45
46     return 0;
47
48  }  // end main
```

```
Exception occurred: runtime_error in function3
```

fig13_03.cpp
(2 of 2)

13.9 建構子, 解構子, 與例外處理

- 建構子內的錯誤
  - **new** 失敗; 不能配置記憶體
  - 程式直接終止,無法呼叫解構子
  - 解決方法:

```
Increment::Increment( int c, int i ) {
   try {
         ary1=new int[c];
         ary2=new int[c];
   }
   catch(...) {
         delete this;
         throw;
   }
} // end constructor Increment
```

13.9 建構子, 解構子, 與例外處理

- 成員函式的錯誤
  - 物件型態
    - MyClass myClass(...);
    - 物件結束前會自動呼叫解構子
  - 物件指標
    - MyClass *myClassPtr=new MyClass(...);
    - 例外情況可能無法delete物件
    - 可將物件指標宣稱成自動指標auto_ptr

13.10 例外與繼承

- 例外類別
  - 可繼承基礎例外類別
    - I.e., exception
  - 用catch捕捉基礎類別也能捕捉衍生類別
    - 多型程式設計

13.11 處理new失敗情形

- 當new失敗時
  - 丟出bad_alloc例外
    - 定義在 <new>
  - 某些編譯器會讓 new 傳回 0
  - 結果跟編譯器有關

Outline

fig13_04.cpp
(1 of 2)

```
1  // Fig. 13.4: fig13_04.cpp
2  // 範例程式
3  // 說明new發生錯誤時會傳回0
4  #include <iostream>
5
6  using std::cout;
7
8  int main()
9  {
10    double *ptr[ 50 ];
11
12    // 配置記憶體給 ptr
13    for ( int i = 0; i < 50; i++ ) {
14       ptr[ i ] = new double[ 5000000 ];
15
16       // 配置記憶體失敗new會傳回0
17       if ( ptr[ i ] == 0 ) {
18          cout << "Memory allocation failed for ptr[ "
19               << i << " ]\n";
20
21          break;
22
23       } // end if
24
```

Outline

```
25         // 成功配置記憶體
26         else
27            cout << "Allocated 5000000 doubles in ptr[ "
28                 << i << " ]\n";
29
30      } // end for
31
32      return 0;
33
34   }  // end main
```

```
Allocated 5000000 doubles in ptr[ 0 ]
Allocated 5000000 doubles in ptr[ 1 ]
Allocated 5000000 doubles in ptr[ 2 ]
Allocated 5000000 doubles in ptr[ 3 ]
Memory allocation failed for ptr[ 4 ]
```

fig13_04.cpp
(2 of 2)

fig13_04.cpp
output (1 of 1)

Outline

fig13_05.cpp
(1 of 2)

```
1  // Fig. 13.5: fig13_05.cpp
2  // 範例程式-使用標準new函式
3  // 說明 new 失敗會丟出 bad_alloc 例外
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8
9  #include <new> // 引用new標頭檔
10
11 using std::bad_alloc; // 使用bad_alloc例外
12
13 int main()
14 {
15    double *ptr[ 50 ];
16
17    // 嘗試配置記憶體
18    try {
19
20       // 配置記憶體給 ptr[ i ]; new throws bad_alloc
21       // 失敗:new 會丟出bad_alloc例外
22       for ( int i = 0; i < 50; i++ ) {
23          ptr[ i ] = new double[ 5000000 ];
24          cout << "Allocated 5000000 doubles in ptr[ "
25               << i << " ]\n";
26       }
27
28    } // end try
```

Outline

```
29
30    // 處理 bad_alloc 例外
31    catch ( bad_alloc &memoryAllocationException ) {
32       cout << "Exception occurred: "
33            << memoryAllocationException.what() << endl;
34
35    } // end catch
36
37    return 0;
38
39 }  // end main
```

```
Allocated 5000000 doubles in ptr[ 0 ]
Allocated 5000000 doubles in ptr[ 1 ]
Allocated 5000000 doubles in ptr[ 2 ]
Allocated 5000000 doubles in ptr[ 3 ]
Exception occurred: Allocation Failure
```

fig13_05.cpp
(2 of 2)

fig13_05.cpp
output (1 of 1)

13.11 處理new失敗情形

- **set_new_handler**
  - 標頭檔 **<new>**
  - 設定一個函式處理new的錯誤
    - 沒有輸入參數
    - 傳回值:void
  - 輸入參數:函式指標
  - 設定完, new失敗會呼叫此函式,不丟出例外
- 範例程式

Outline

fig13_06.cpp
(1 of 2)

```
1  // Fig. 13.6: fig13_06.cpp
2  // set_new_handler使用範例
3  #include <iostream>
4
5  using std::cout;
6  using std::cerr;
7
8  #include <new>     // 引用new
9
10 using std::set_new_handler;
11
12 #include <cstdlib> // 引用 abort 函式原型
13
14 void customNewHandler()
15 {
16    cerr << "customNewHandler was called";
17    abort();
18 }
19
20 // 使用 set_new_handler 處理記憶體配置失敗
21 int main()
22 {
23    double *ptr[ 50 ];
24
```

Outline

```
25     // 指定 customNewHandler 函式,
26     // 處理記憶體配置失敗
27     set_new_handler( customNewHandler );
28
29     // 配置記憶體給 ptr[ i ]
30     // 記憶體配置失敗時customNewHandler會被呼叫
31     for ( int i = 0; i < 50; i++ ) {
32        ptr[ i ] = new double[ 5000000 ];
33
34        cout << "Allocated 5000000 doubles in ptr[ "
35             << i << " ]\n";
36
37     } // end for
38
39     return 0;
40
41  }  // end main
```

```
Allocated 5000000 doubles in ptr[ 0 ]
Allocated 5000000 doubles in ptr[ 1 ]
Allocated 5000000 doubles in ptr[ 2 ]
Allocated 5000000 doubles in ptr[ 3 ]
customNewHandler was called
```

fig13_06.cpp
(2 of 2)

fig13_06.cpp
output (1 of 1)

13.12 auto_ptr類別與動態記憶體配置

- 宣稱指標, 使用new配置記憶體
  - 例外發生在delete之前會導致 - Memory leak
- 類別樣版 auto_ptr
  - 標頭檔 <memory>
  - 當指標超出範圍, 自動呼叫delete
- 用法

```
auto_ptr< MyClass > newPointer( new MyClass() );
```

  - newPointer會指到動態配置的物件

Outline

fig13_07.cpp
(1 of 3)

```
1  // Fig. 13.7: fig13_07.cpp
2  // auto_ptr 使用範例
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <memory> // 使用auto_ptr
9
10 using std::auto_ptr; // 使用auto_ptr類別定義
11
12 class Integer {
13
14 public:
15
16    // 建構子
17    Integer( int i = 0 )
18       : value( i )
19    {
20       cout << "Constructor for Integer " << value << endl;
21
22    } // end Integer constructor
23
```

Outline

```
24    // 解構子
25    ~Integer()
26    {
27       cout << "Destructor for Integer " << value << endl;
28
29    } // end Integer destructor
30
31    // 設定value的值
32    void setInteger( int i )
33    {
34       value = i;
35
36    } // end function setInteger
37
38    // 傳回value的值
39    int getInteger() const
40    {
41       return value;
42
43    } // end function getInteger
44
45 private:
46    int value;
47
48 };  // end class Integer
49
```

fig13_07.cpp
(2 of 3)

Outline

```
50 // 使用auto_ptr操作Integer物件
51 int main()
52 {
53    cout << "Creating an auto_ptr object that points to an "
54         << "Integer\n";
55
56    // 令 auto_ptr 指到 Integer 物件
57    auto_ptr< Integer > ptrToInteger( new Integer( 7 ) );
58
59   cout << "\nUsing the auto_ptr to manipulate the Integer\n";
60
61    // 使用 auto_ptr 設定 Integer 的值
62    ptrToInteger->setInteger( 99 );
63
64    // 使用 auto_ptr 取得 Integer 的值
65    cout << "Integer after setInteger: "
66         << ( *ptrToInteger ).getInteger()
67         << "\n\nTerminating program" << endl;
68
69    return 0;
70
71 }  // end main
```

fig13_07.cpp
(3 of 3)

Outline

```
Creating an auto_ptr object that points to an Integer
Constructor for Integer 7

Using the auto_ptr to manipulate the Integer
Integer after setInteger: 99

Terminating program
Destructor for Integer 99
```

fig13_07.cpp
output (1 of 1)