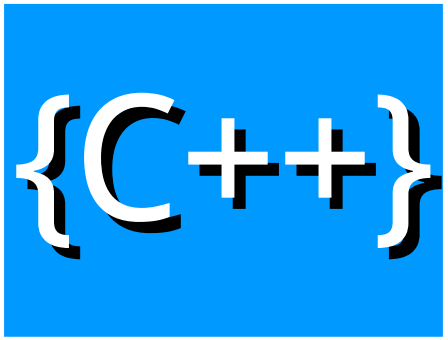
Exception

Week 11



Yang-Cheng Chang
Yuan-Ze University
yczhang@saturn.yzu.edu.tw

Exception(異常)

- 異常指的是所有可以造成電腦無法正常處理的狀況
- 經常遇到的異常狀況可以分爲五種：
  - 資源不足： 又分爲「記憶體不足」 (out of memory) 和「儲存空間耗盡」兩種情況。
  - 開檔失敗： 又分爲「讀檔」和「存檔」兩種情況。
  - 越界： 索引超過上限或低於下限。
  - 除數爲零： 在進行數值除法運算時，經常造成當機。
  - 使用者輸入錯誤

當異常發生的可能結果

- 無預警地突然結束或當機。
- 程式自動結束，但沒有任何相關訊息。
- 發出警告訊息後程式自行正常結束。
- 自行跳過異常部份，但其後的運算可能沒有意義。
- 通知使用者異常發生的種類，並引導使用者逐步排除異常後繼續工作。

傳統異常的處理方式

- 一般程式碼與異常處理程式碼交錯混合
- 難於閱讀，修改與除錯

Pseudocode outline

```
Perform a task
If the preceding task did not execute correctly
   Perform error processing
Perform next task
If the preceding task did not execute correctly
   Perform error processing
…
```

C++ 提供的異常基本語法

```
1 #include <iostream>
2 using namespace std;
3 int main ()
4 {
5     try
6     {
7         throw 10;
8     }
9     catch (int e)
10    {
11        cout  << “We have a problem!!” << endl;
12    }
13    return 0;
14 }
```

C++ 提供了 try (嚐試) ， throw (丟擲) ，和 catch (捕捉) 三個關鍵字。
把所有可能發生異常的敘述都放在「 try 區塊」 (try block) 。

Throw

- 在所有可能發生異常的敘述之前，都可以使用「throw 敘述」丟擲出一個異常訊息物件 (the exception message object)。
- 通常「throw 敘述」都不直接在「try 區塊」之內，而是在「try 區塊所呼叫的函數」內。

```
1 #include <iostream>
2 using namespace std;
3
4 void maybeError(int x)
5 {
6     if ( x >= 100){
7         // code
8         throw 10;
9     }
10 }
```

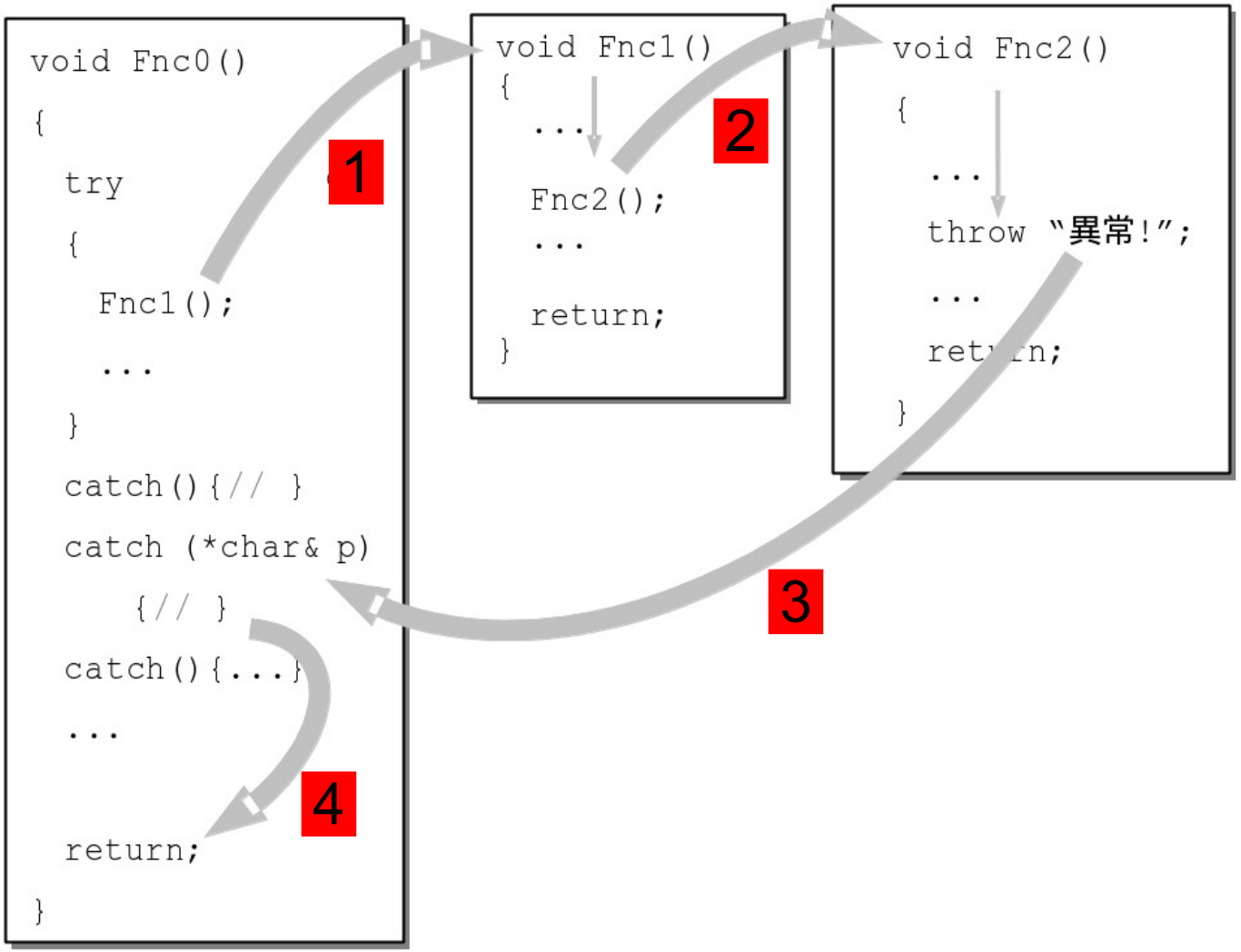
```
11
12 int main(int argc, const char *argv[])
13 {
14     int x = 100, y = 0;
15     try {
16         // code
17         maybeError(x);
18         if (y < 0){
19             //code
20             throw 20;
21         }
22     }
23     catch(int param){
24         cout << " x >= 100 " << endl;;
25     }
26
27     return 0;
28 }
```

catch{}

- 「catch 區塊」用來承接由相關「throw 敘述」所丟出來的訊息物件
- 在所有的「catch 區塊」之後加上一個捕捉所有異常物件之區塊 (catch-all block)，其中三個點「...」代表「所有可能」的意思。

```
try {
  // code here
}
catch (int param) {
   cout << "int exception";
}
catch (char param) {
   cout << "char exception";
}
catch (...) {
   cout << "default exception";
}
```

異常的處理過程


void Fnc0()
{
try
{
Fnc1();
...
}
catch(){// }
catch (*char& p)
{// }
catch(){...}
...
return;
}
1
void Fnc1()
{
...
Fnc2();
...
return;
}
2
void Fnc2()
{
...
throw "異常!";
...
return;
}
3
4

Example: Handling an Attempt to Divide by Zero

```
// Fig. 27.1: DivideByZeroException.h
// Class DivideByZeroException definition.
#include <stdexcept> // stdexcept header file contains runtime_error
using std::runtime_error; // standard C++ library class runtime_error

// DivideByZeroException objects should be thrown by functions
// upon detecting division-by-zero exceptions
class DivideByZeroException : public runtime_error
{
public:
   // constructor specifies default error message
   DivideByZeroException::DivideByZeroException()
      : runtime_error( "attempted to divide by zero" ) {}
}; // end class DivideByZeroException
```

```cpp
// Fig. 27.2: Fig27_02.cpp
// A simple exception-handling example that checks for
// divide-by-zero exceptions.
#include <iostream>
using std::cin;
using std::cout;
using std::endl;

#include "DivideByZeroException.h" // DivideByZeroException class

// perform division and throw DivideByZeroException object if
// divide-by-zero exception occurs
double quotient( int numerator, int denominator )
{
   // throw DivideByZeroException if trying to divide by zero
   if ( denominator == 0 )
      throw DivideByZeroException(); // terminate function

   // return division result
   return static_cast< double >( numerator ) / denominator;
} // end function quotient

int main()
{
   int number1; // user-specified numerator
   int number2; // user-specified denominator
   double result; // result of division

   cout << "Enter two integers (end-of-file to end): ";
```

```

   // enable user to enter two integers to divide
   while ( cin >> number1 >> number2 )
   {
      // try block contains code that might throw exception
      // and code that should not execute if an exception occurs
      try
      {
         result = quotient( number1, number2 );
         cout << "The quotient is: " << result << endl;
      } // end try

      // exception handler handles a divide-by-zero exception
      catch ( DivideByZeroException &divideByZeroException )
      {
         cout << "Exception occurred: "
            << divideByZeroException.what() << endl;
      } // end catch

      cout << "\nEnter two integers (end-of-file to end): ";
   } // end while

   cout << endl;
   return 0; // terminate normally
} // end main
```

Example: std::exception::what

```cpp
#include <exception>
#include <iostream>
#include <string>

class Exception : public std::exception
{
    std::string _msg;
public:
    Exception(const std::string& msg) : _msg(msg){}

    virtual const char* what() const noexcept override
    {
        return _msg.c_str();
    }
};

int main()
{
    try
    {
        throw Exception("Something went wrong...\n");
    }
    catch(Exception& e)
    {
        std::cout << e.what() << std::endl;
    }
}
```

```cpp
// exception::what
#include <iostream>       // std::cout
#include <exception>      // std::exception

struct ooops : std::exception {
  const char* what() const noexcept {return "Ooops!\n";}
};

int main ()
{
  try {
      throw ooops();
  }
  catch (std::exception& ex) {
      std::cout << ex.what();
  }
  return 0;
}
```

Assignment 11

- Chinese Year

  能夠處理中國的生肖年，產生生肖年與西元年的對應

  – 有三種建構函式，範例如下

    - ChineseYear("Tiger")

      產生過去距離今年 (2019) 最近的虎年

      std::cout 輸出爲 Tiger 2010-2011

    - ChineseYear(2015)

      以西元年產生生肖年

      std::cout 輸出爲 Goat 2015-2016

    - ChineseYear()

      預設以今年 (2019) 產生生肖年

      std::cout 輸出爲 Pig 2019-2020

  – 可以用 std::cout 輸出顯示生肖年，格式如下

    Tiger 2010-2011

Assignment 11

- 可以用加法改變生肖年，例如：

```
cy1+=1
cy2+3
5+cy2
```

- 可以檢查出兩種例外，採用 try / catch 型式，當發生例外時 依照所發生的例外分別 throw 對應的例外類別，並在對應的 catch 內顯示例外的錯誤訊息，這兩種例外類別都繼承自 std::exception，透過覆寫 std::exception:what() 定義自己的錯誤訊息
  - 需要定義的例外類別 ChineseYear::WrongGregorianYear 顯示錯誤訊息 "Invalid Gregorian Year (must be >= 1900)"
  - 需要定義的例外類別 ChineseYear::WrongChineseYear 顯示錯誤訊息 "Invalid Chinese Year"

Convert Chinese Year to Gregorian Year

子 鼠	丑 牛	寅 虎	卯 兔	辰 龍	巳 蛇	午 馬	未 羊	申 猴	酉 雞	戌 狗	亥 豬
1900	1901	1902	1903	1904	1905	1906	1907	1908	1909	1910	1911
1912	1913	1914	1915	1916	1917	1918	1919	1920	1921	1922	1923
1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947
1948	1949	1950	1951	1952	1953	1954	1955	1956	1957	1958	1959
1960	1961	1962	1963	1964	1965	1966	1967	1968	1969	1970	1971
1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995
1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007
2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019
2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043
2044	2045	2046	2047	2048	2049	2050	2051	2052	2053	2054	2055
2056	2057	2058	2059	2060	2061	2062	2063	2064	2065	2066	2067
2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091
2092	2093	2094	2095	2096	2097	2098	2099	2100	2101	2102	2103

鼠 Rat
牛 Ox
虎 Tiger
兔 Rabbit
龍 Dragon
蛇 Snake
馬 Horse
羊 Goat
猴 Monkey
雞 Rooster
狗 Dog
豬 Pig

Base year: 2008 Rat 2008-2009
Earliest year: 1900 Rat 1900-1901
Default year: 2019 Pig 2019-2020

The output of main.cpp

```
Trying the year of 1900
Invalid Gregorian Year (must be >= 1900)
Trying the year of the Panda
Invalid Chinese Year
cy1 = Tiger 2010-2011, cy2 = Goat 2015-2016, cy3 = Pig 2019-2020
cy1 = Rabbit 2011-2012
cy2 + 3 = Dog 2018-2019
5 + cy2 = Rat 2020-2021
```

main.cpp

```cpp
#include <iostream>
#include "ChineseYear.h"
using namespace std;
int main()
{
    // Test 1: Create a new Chinese Year with an animal
    ChineseYear cy1 = ChineseYear("Tiger");
    // Test 2: Create a new Chinese Year with a year
    ChineseYear cy2 = ChineseYear(2015);
    // Test 3: Create a default Chinese Year
    ChineseYear cy3;
    // Test 4: Create a default Chinese Year
    cout << "Trying the year of 1900" << endl;
    try {
        ChineseYear cy3 = ChineseYear(1899);
    }
    catch (ChineseYear::WrongGregorianYear &e) {
        cerr << e.what() << endl;
    }
    // Test 5: Create a wrong Chinese Year
    cout << "Trying the year of the Panda" << endl;
    try {
        ChineseYear cy3 = ChineseYear("Panda");
    }
    catch (ChineseYear::WrongChineseYear &e) {
        cerr << e.what() << endl;
    }
    // Test 6: Display cy1, cy2 and cy3
    cout << "cy1 = " << cy1 << ", cy2 = " << cy2 << ", cy3 = "<< cy3 << endl;
    // Test 7: Increment cy1 and display
    cy1 += 1;
    cout << "cy1 = " << cy1 << endl;
    // Test 8: Add 3 to cy2 and display
    cout << "cy2 + 3 = " << cy2  + 3 << endl;
    // Test 9: Add 5 to cy2 and display, different order
    cout << "5 + cy2 = " << 5 + cy2 << endl;
    return 0;
}
```