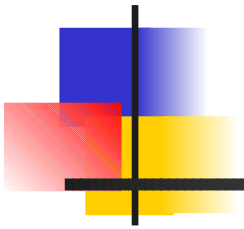
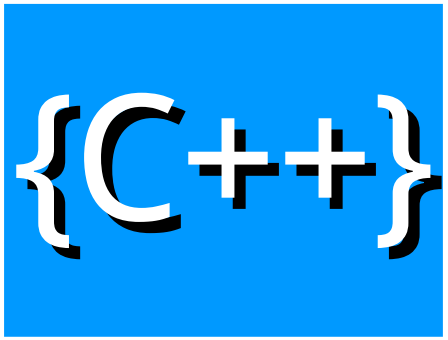


Template

Week 8



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

繼承 C++ Template Class 的注意事項

```
template< typename T>
class A
{
protected:
    T m_data;
};

template< typename T>
class B : public A< T>
{
public:
    void Test(T t)
    {
        m_data = t;


    }
};

int main(int argc, char** argv)
{
    B< int> b;
    b.Test(1);
}
```

編譯錯誤

```
example-wrong.cpp: In member function
'void B<T>::Test(T)':
example-wrong.cpp:14:9: error: 'm_data'
was not declared in this scope
```

正確的做法

方法一

```
void Test(T t)
{
    A< T>::m_data = t;
}
```

方法二

```
void Test(T t)
{
    this->m_data = t;
}
```

函式樣板的實例化

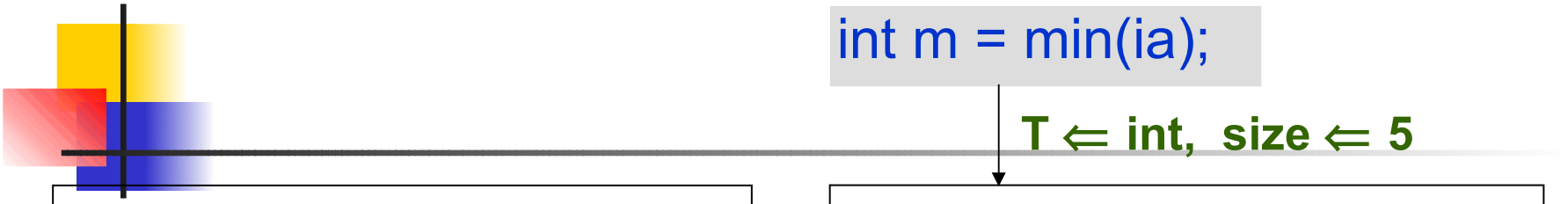
- 當呼叫函式樣板時，C++ 編譯器會依據引數的資料型態，自動產生出所需的函式，這個過程稱爲實例化（instantiate）。

```
template <typename T>
T min (T x, T y)
{
    return (x < y)? x : y
}
```

```
int m, x, y;
m = min(x, y);
```

T ⇐ int

```
int min (int x, int y)
{
    return (x < y)? x : y
}
```



```
template <typename T, int size>
T min (const T (&a)[size])
{
        T min_val = a[0];
        for (int i = 0; i < size; i++)
                if (a[i] < min_val)
                        min_val = a[i];
        return min_val;
}
```

```
int ia[] = { 5, 2, 8, 3, 9};
int m = min(ia);
```

T ⇐ int, size ⇐ 5

```
int min (const int (&a)[5])
{
        …
}
```

```
double da[] = { 5.0, 2.0, 8.3};
double m = min(da);
```

T ⇐ double, size ⇐ 3

```
double min (const double (&a)[3])
{
        …
}
```

顯式實例化宣告

- 在前面的兩種函式樣板編譯模式中，函式樣板只有在被呼叫後才會產生眞實的函式。
- 但在某些情況下，即使函式樣板沒有被呼叫也需要產生眞實的函式（如：製作程式庫）。這時我們可以利用 explicit instantiation 的宣告。

```
template <typename T>
T sum (T op1, int op2) { … }

// explicit instantiation declaration

template int* sum<int *> (int *, int);
```

產生　int* sum(int *, int);

C++ Template 的宣告與定義分離的做法(一)

class.h

```
template <typename T>
class demo
{
        public:
                demo(T v){ value = v;}
                void set(T v);
        private:
                T value;
};
```

class.cpp

```
#include "class.h"

template <typename T>
void demo<T>::set(T v)
{
        value = v;
}
template class demo<int>;
template class demo<float>;
```

Explicit instantiatation
(顯式實例化)

main.cpp

```
#include "class.h"
int main()
{
        demo<int> intdemo;
        demo<float> floatdemo;
}
```

C++ Template 的宣告與定義分離的做法(二)

class.h

```
template <typename T>
class demo
{
        public:
                demo(T v){ value = v;}
                void set(T v);
        private:
                T value;
};
#include "class.cpp"
```

class.cpp

```
#include "class.h"

template <typename T>
void demo<T>::set(T v)
{
        value = v;
}
```

與不分離的做法本質上相同

main.cpp

```
#include "class.h"
int main()
{
        demo<int> intdemo;
        demo<float> floatdemo;
}
```

C++ Template 的宣告與定義分離的做法(三)

class.h

```
template <typename T>
class demo
{
        public:
                demo(T v){ value = v;}
                void set(T v);
        private:
                T value;
};
```

class.cpp

```
#include "class.h"

export template <typename T>
void demo<T>::set(T v)
{
        value = v;
}
```

編譯器支援度不高

main.cpp

```
#include "class.h"
int main()
{
        demo<int> intdemo;
        demo<float> floatdemo;
}
```

Template Explicit Specialization

- 當函式樣板對某些資料型態不適用或效率不佳時，我們可以設計一個特殊化的函式樣板來取代，稱爲顯式特例化 (Explicit Specialization)
- 舉例來說，底下的函式樣板無法用來獲得較小的 C 字串

```
template <typename T>
T min (T x, T y) { return (x < y)? x : y }
```

- 因此我們就寫一個特殊化的函式樣板如下：

```
template<> const char * min (const char *x, const char * y)
{ return strcmp(x,y) < 0? x : y; }
```

Template Explicit Specialization

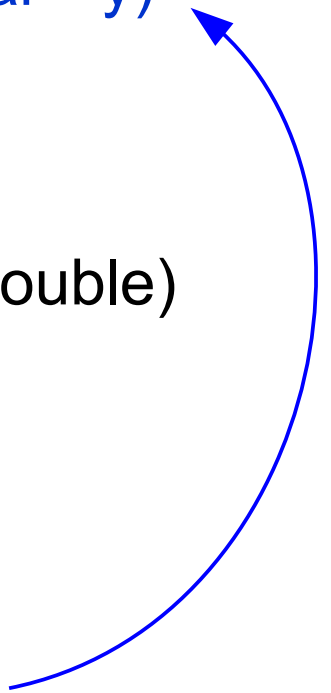
```
template <typename T>
T min (T x, T y) { return (x < y)? x : y }

template<> const char * min (const char *x, const char * y)
    { return strcmp(x,y) < 0? x : y; }

int d = min(2.0, 3.0);    // 產生  double min(double, double)

int m = min(2, 3);        // 產生  int min(int , int )

// 產生  const char * min(const char *, const char *)
const char *cp = min("book", "apple");
```



Assignment 8

- 使用 assignment 7 的 template class Array 來完成 class SortedArray
- class SortedArray 必須符合以下要求
  - 繼承自 template class Array
  - 必須是 template class
  - SortedArray 中的元素的必須經過排序
    - 數值類的型別 (int, double, float) 由大到小來排序
    - string 則以字串的第一個字元來排序，不分大小寫，以反向字母順序來排序，例如 : "zoom" > "Word" > "and"
  - 新增一個成員函式 addValue(T t) ，用來加入新元素
    - 必須確保加入新元素後，所有的元素必須依照規則排序

Assignment 8

- 需有一個 template 函式，用來比較大小
  - 針對 string 型別進行特例化 (specialization)
  - template<typename T> bool isgreater(T left, T right);

```
template<typename T> bool isgreater(T left, T right);
```