C++ Programming

Chapter 13 Templates 模板

Zheng Guibin
（郑贵滨）

13.1 Introduction 引言

13.1 引言

- *模板是生成类或函数的框架。*
- 与类或函数显式指定数据类型不同，模板使用形参。
- 当实际数据类型赋值给形参的时候，才由编译器生成类或函数。

13.2函数模板（ Function templates ）

- 寻找两个整型值的最大值的函数

```
int maximun ( const int n1,const int n2 )
{
   if (n1>n2)
      return n1;
   else
       return n2;
}
```

- 寻找两个浮点值的最大值的函数——一个几乎相同的函数

```
float maximun ( const float n1,const float n2 )
{
   if (n1>n2)
      return n1;
   else
       return n2;
}
```

需要两个不同函数的原因在于两个函数中的函数头不一样。

13.2 函数模板（Function templates）

- 函数模板允许这样的两个函数被一个单独的函数取代，在这个单独的函数里，形参数据类型用T（T表示一种类型）表示，从而提供一个泛型或类型无关的函数，它适用于所有的数据类型。
- T叫做类型形参，通常用T来表示，也可以用其他字母表示。
- **Program Example P13A**

```
6    template <typename T>
7    T maximum( const T n1, const T n2 )
8    {
9       if ( n1>n2)
10         return n1;
11      else
12         return n2;
13   }
```

13.2 函数模板（Function templates）

- *函数模板的声明由关键字**template*** 和包含一个或多个数据类型形参的参数列表构成。
- 形参数据类型前面可以是关键字***class***，也可以是意义更加明确的关键字***typename***。
- 类型形参T要用尖括号<和>括起来。当同时使用多种数据类型的形参时，他们之间要用逗号分开。

```
6    template <typename T>
7    T maximum( const T n1, const T n2 )
8     {
9        if ( n1>n2)
10           return n1;
11       else
12           return n2;
13    }
```

```
main()
{
  char c1 = 'a', c2 = 'b' ;
  int i1 = 1, i2 = 2 ;
  float f1 = 2.5, f2 = 3.5 ;

  cout << maximum( c1, c2 ) << endl ; //  a maximum() for chars
  cout << maximum( i1, i2 ) << endl ; //  a maximum() for ints
  cout << maximum( f1, f2 ) << endl ; //  a maximum() for floats
}
```

13.2函数模板（Function templates）

```
15 main()
16 {
17   char c1 = 'a', c2 = 'b' ;
18   int i1 = 1, i2 = 2 ;
19   float f1 = 2.5, f2 = 3.5 ;
20
21   cout << maximum( c1, c2 ) << endl ; //  a maximum() for chars
22   cout << maximum( i1, i2 ) << endl ; //  a maximum() for ints
23   cout << maximum( f1, f2 ) << endl ; //  a maximum() for floats
24 }
```

第21行,编译器生成函数原型:用参数c1和c2的数据类型即char代替第6行的类型参数T。。

这个过程称之为模板的实例化，其结果是由编译器产生的一个常规函数

13.2函数模板（Function templates）

- 在上面的程序中，没有使用3个独立的函数，取而代之的是一个单独的函数模板。
  - 程序的源文件变小， 但是编译器利用模板生成了三个不同的函数，所以可执行文件的大小仍然是不变的。
- 模板的定义可以放在头文件里。

```
// Program example P13B
// function template.
#include <iostream>
#include "maximum.h"
using namespace std ;
main()
{
  char c1 = 'a', c2 = 'b' ;
  int i1 = 1, i2 = 2 ;
  float f1 = 2.5, f2 = 3.5 ;

  cout << maximum( c1, c2 ) << endl ;
  cout << maximum( i1, i2 ) << endl ;
  cout << maximum( f1, f2 ) << endl ;
}
```

13.2函数模板（Function templates）

- 利用函数模板对数组元素进行排序
- 程序例 P13C
  - 这个程序在头文件*bubble.h, swap.h* 和 *show.h*中使用了三个函数模板。

13.3类模板（Class templates）

- 模板不仅能生成函数，也能用来生成完整的C++类。

  模板允许一个类用于所有数据类型

- 类模板也称为参数化类型

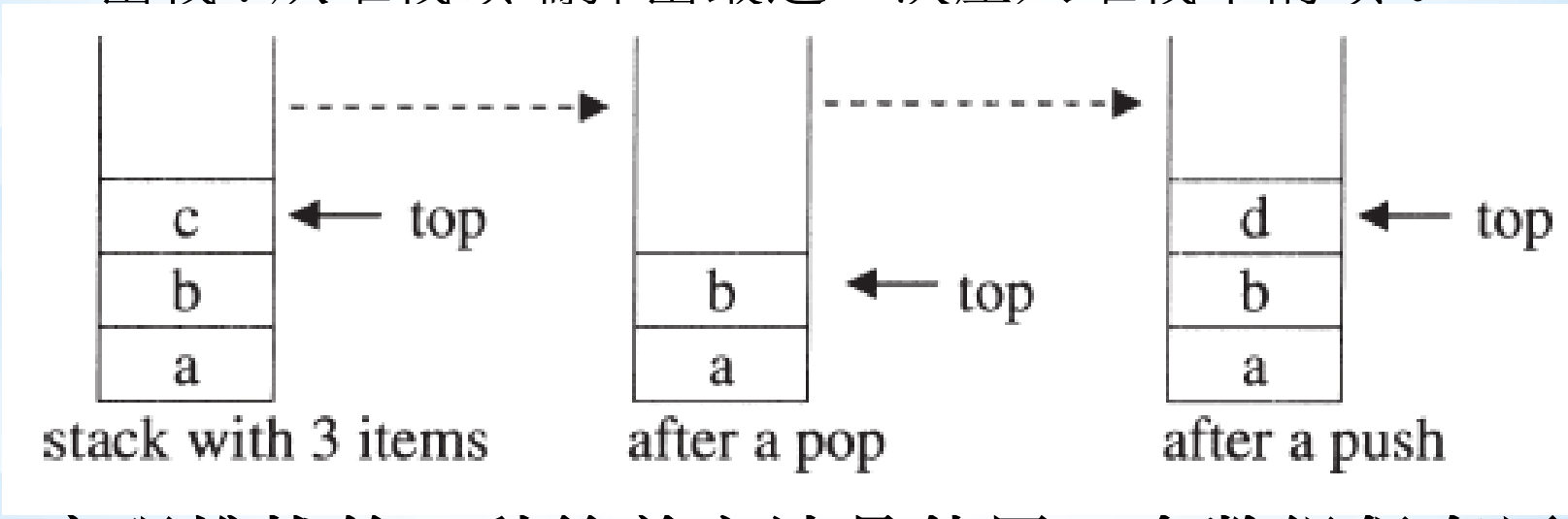
- 参数化类型是由其他数据类型甚至是一些未指定的数据类型来定义的数据类型。

- 堆栈数据结构的操作

  可以用对一摞盘子的操作来形容对一个堆栈数据结构的操作，放盘子和取盘子只能在这一摞盘子的顶部进行。

13.3类模板（Class templates）

- 一个堆栈数据结构主要包含两种主要的操作：
  - 压栈: 向堆栈顶端添加一项。
  - 出栈：从堆栈顶端弹出最近一次压入堆栈中的项 。



- 实现堆栈的一种简单方法是使用一个数组保存压入堆栈中的项，使用一个变量记录数组“顶端”的下标值。

13.3 Class templates

- 堆栈可以定义为类模板，使用类型形参来指定要存储的项的数据类型，使用一个整形变量来指示当前的栈顶。

- 堆栈类模板的定义保存在头文件 *stack.h* 里。

```
// Declaration of stack class template.
#if !defined STACK_T_H
#define STACK_T_H
```

13.3类模板（Class templates）

- 引用类模板必须包含它的形参列表。
  - 成员函数的定义使用stack <T> 而不是仅使用 stack, 如果堆栈类不是模板，则仅使用stack。
  - 程序例 **P13D**

```
#include <iostream>
#include "stack.h"
using namespace std ;
main(){
  stack <int> i_stack ;       // A st
  stack <char> c_stack( 5 ) ; // A
  int i, n, int_data ;
  char char_data ;
  bool success ;
```

- 堆栈数据类型由两个尖括号 < 和> 之间的数据类型指定。
- 第9行用类模板生成了一个堆栈类，来处理10个整形数据项。
- 第10行用类模板生成了一个堆栈类，来处理5个字符数据项。
- 严格来讲，从类模板生成的类，称为模板类。

```
// Push some values onto c_stack.
  c_stack.push( 'a' ) ;
  c_stack.push( 'b' ) ;
  c_stack.push( 'c' ) ;


  // Use a for loop to fill i_stack.
  n = i_stack.stack_size() ;
  for( i = 0 ; i < n ; i++ )
    i_stack.push( i ) ;

  // Display the values on c_stack.
  cout << "character stack data:" << endl ;
  n = c_stack.number_stacked() ;
  for ( i = 0 ; i < n ; i++ )
{
    success = c_stack.pop( char_data ) ;
    if ( success )
      cout << char_data << ' ' ;
  }
  cout << endl ;
```

```
 // Display the values on i_stack.
  cout << "integer stack data:" << endl ;
  n = i_stack.number_stacked() ;
  for ( i = 0 ; i < n ; i++ )
  {
    success = i_stack.pop ( int_data ) ;
    if ( success )
      cout << int_data << ' ' ;
  }
  cout << endl ;
}
```

13.3类模板（Class templates）

- 为了创建模板，首先要写一个非模板特定数据类型的函数或者类。

- 当这个指定数据类型的函数或者类运行的令人满意后，再用模板形参来取代其中的特定数据类型。

编程陷阱（Programming Pitfalls）

- 1. 不要以为函数原型中的所有运算符都适合作为传递给模板形参的数据类型。

```
point p1(1,2), p2(3,4), p3(0,0);
p3 = maximum(p1,p2);
```

使用函数模板到一个点类的对象中，如果在点类的定义中>没有重载，程序就会出错。

编程陷阱（Programming Pitfalls）

- **2.每一个模板形参前都必须写上关键字typename。**

  template <**typename T1, T2**>

  程序报错因为T2前面也应该加上关键字typename。

- **3. 每一个指定类型的形参都必须在函数中使用。**
  - 程序将会报错因为 T2 没有被使用。

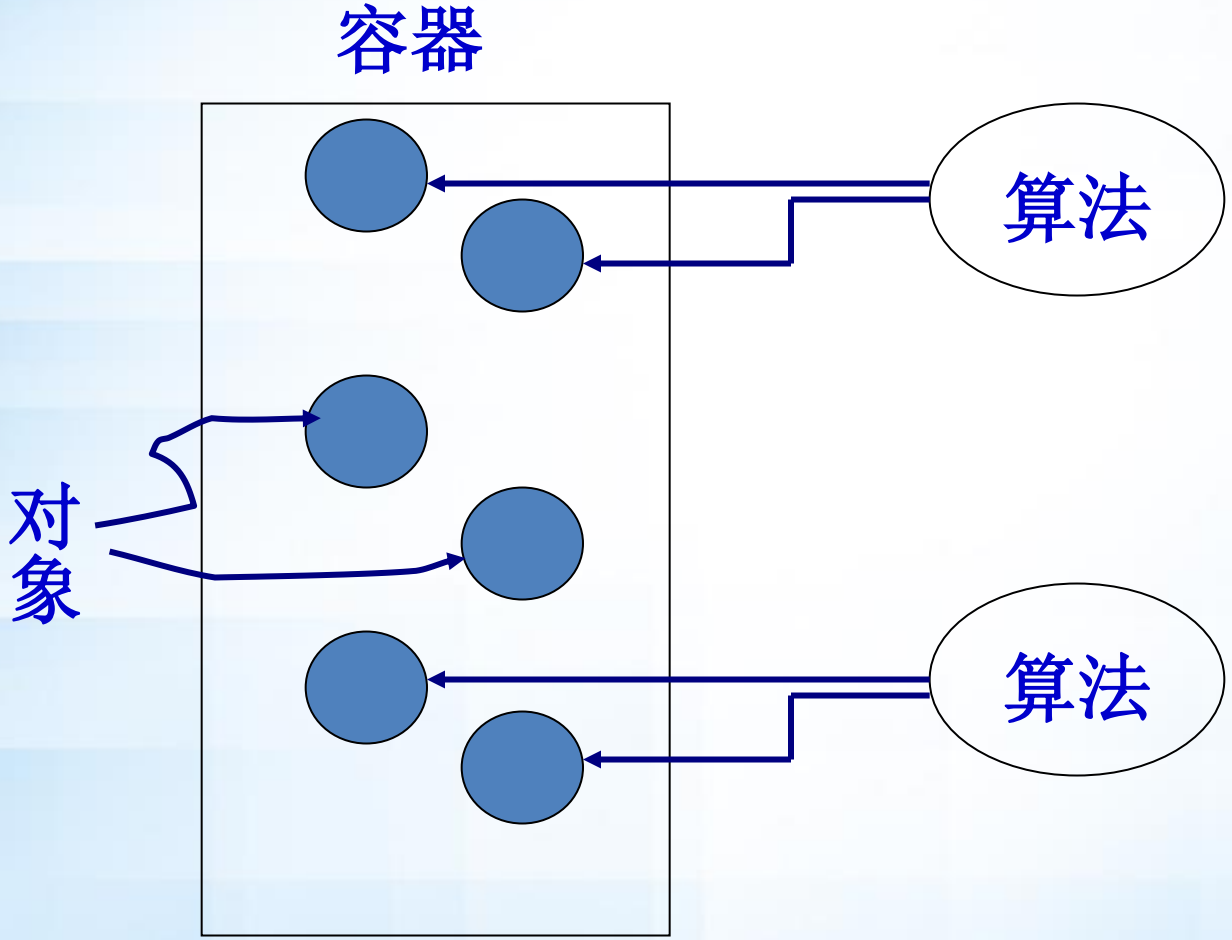
```
template <typename T1, typename T2>
int my_function(T1 var )//Error -  T2 is not used
```

Q & A

Thank You!

标准模板库(STL) 介绍

- STL包括： 容器、迭代器、算法



容器(container)

- 容器是存储数据的一种方式
- 容器分类：

  顺序容器 关联容器和容器适配器
- 顺序容器：

  vector(矢量)、list(链表)、deque(双端队列)
- 关联容器

  set(集合)、multiset、map(映射)、ultimap
- 容器举例：

标准库容器类	说明
顺序容器	
voctor(矢量)	从后面快速插入与删除，直接访问任何元素
list(列表)	从任何地方快速插入与删除，双链表
deque(双端队列)	从前或后面快速插入与删除，直接访问任何元素
关联容器	
set(集合)	快速查找，不允许重复值
multiset(多重集合)	快速查找，允许重复值
map(映射)	一对一映射，基于关键字快速查找，不许重复值
multimap(多重映射)	一对多映射，基于关键字快速查找，允许重复值
容器适配器	
stack(栈)	后进先出**(LIFO)**
queue(队列)	先进先出**(FIFO)**
priority_queue	最高优先级元素总是第一个出列

迭代器(iterator)

- 迭代器类似于指针，用来访问容器中的单个数据项。
- 迭代器由类iterator来表明。
- 不同的迭代器必须用于不同的容器。
- 迭代器分类：

  向前迭代器，双向迭代器，随机迭代器
- 使用迭代器：

  数据访问 与 数据插入

标准库迭代子类型	说明
输入 **InputIterator**	从容器中读取元素。输入迭代子只能一次一个元素地向前移动(即从容器开头到容器末尾)。 要重读必须从头开始。
输出 **OutputIterator**	向容器写入元素 。 输出迭代子只能一次一个元素地向前移动。 输出迭代子要重写，必须从头开始
正向 **ForwardIterator**	组合输入迭代子和输出迭代子的功能， 并保留在容器中的位置(作为状态信息)， 所以重新读写不必从头开始。
双向 **BidirectionalIterator**	组合正向迭代子功能与逆向移动功能 (即从容器序列末尾到容器序列开头)
随机访问 **RandomAccseeIterator**	组合双向迭代子的功能，并能直接访问容器中的任意元素，即可向前或向后调任意个元素。

算法(algorithm)

- STL算法在数据集上进行操作，泛型算法不依赖于具体的容器。
- 泛型算法分类为

(1)不修改序列的操作

find( )、cout( )、equal( )、mismatch( )和search( )等。

(2)修改序列的操作

swap( )、copy( )、transform( )、replace( )、remove( )、reverse( )、rotate( )和fill( )等。

(3)排序、合并和相关的操作

sort( )、binary_search( )、merge( )、min( )和max( )等。