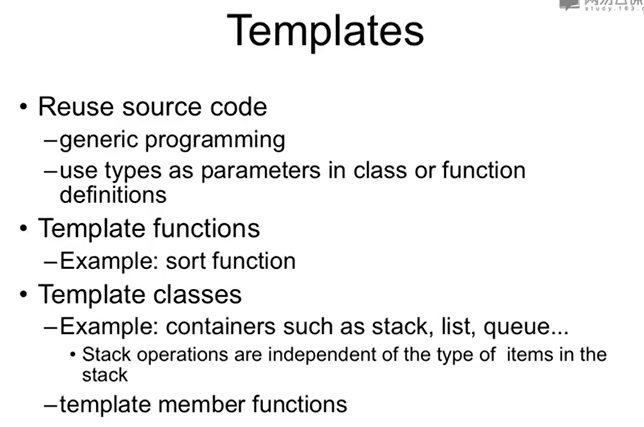


模板适用于对象的处理代码几乎都是一样的，区别于其处理的对象类型。

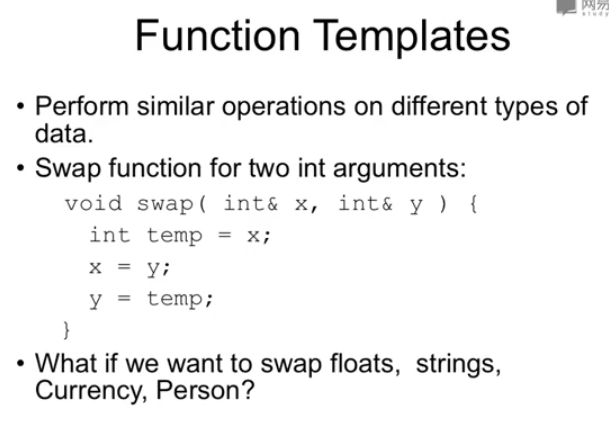
针对这种情况，可以

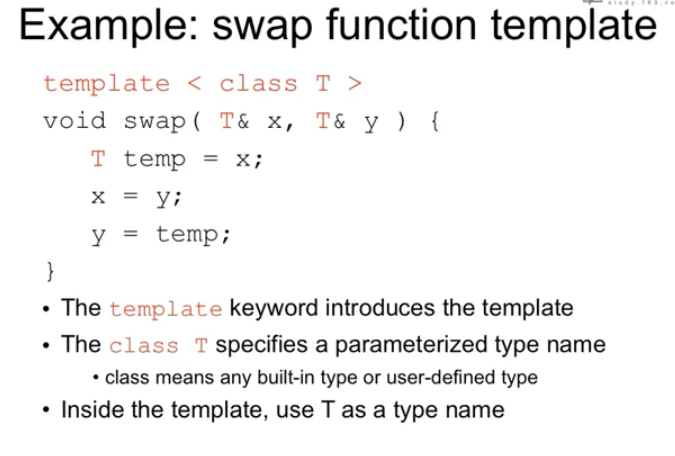
* 使用共同的基类
* 克隆代码
* 将类型设置为auto
  + type unsafe



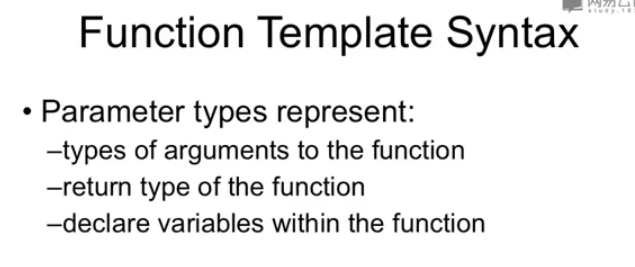
函数模板 & 类模板 & template function template class

* 重用源代码
  + 一般编程（泛型编程）
  + 在类和函数的定义中将 类型types 作为参数

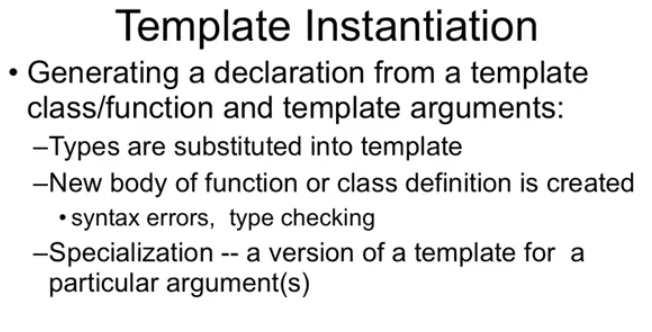




1. template 关键字引出 模板
2. class T 指定参数化的类型名
   1. 可以是内建也可以是自定义类型
3. 模板内部， T视为一个类型名进行使用。



* 函数**参数**类型
* 函数**返回值**类型
* 函数内 可声明**局部变量之类型**

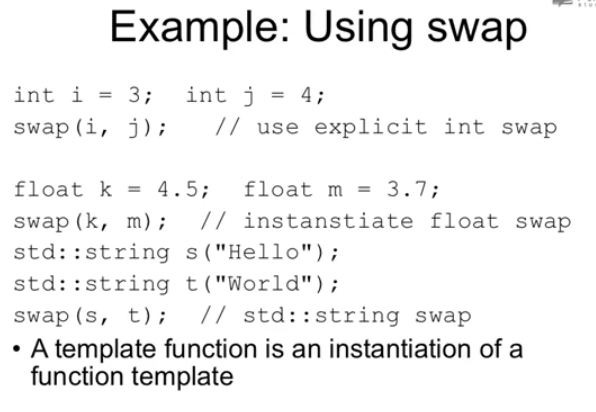


模板函数

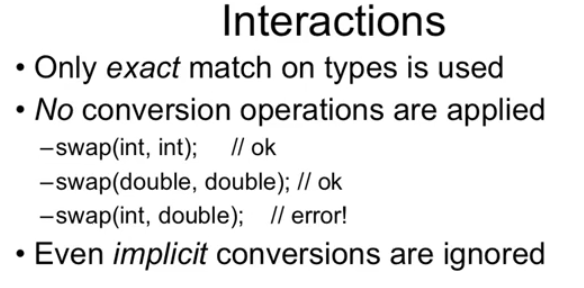
【**声明：declaration，定义：definition**】

模板是一个声明，不是定义。

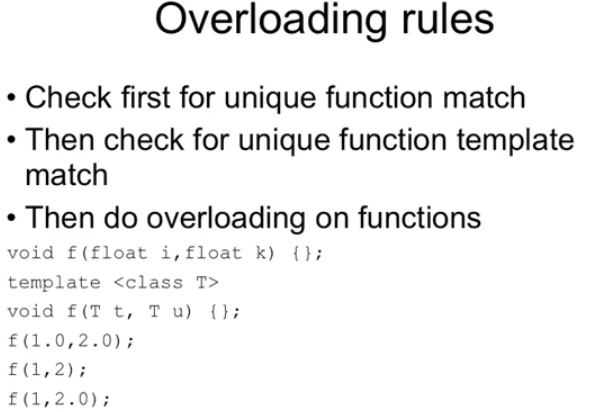
对于参数为float的swap函数，则会初始化一个 float swap函数，这个函数接受两个float类型的函数。



模板函数是函数模板的一个实例instance.

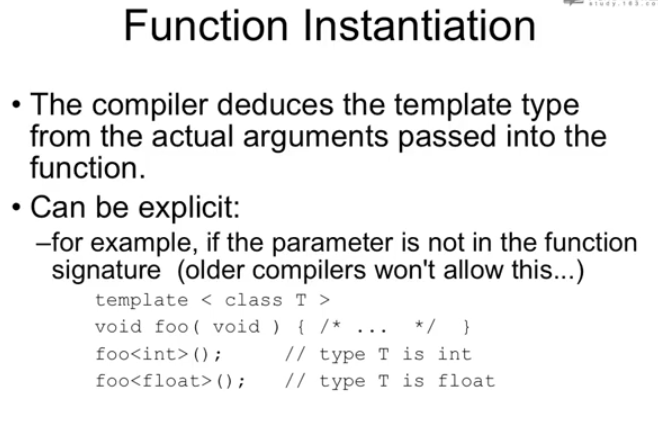


函数模板



重载规则

1. 单独的匹配 **函数**
2. 单独的匹配 **函数模板**
3. 再对函数做**重载** overloading, same name with different signature.



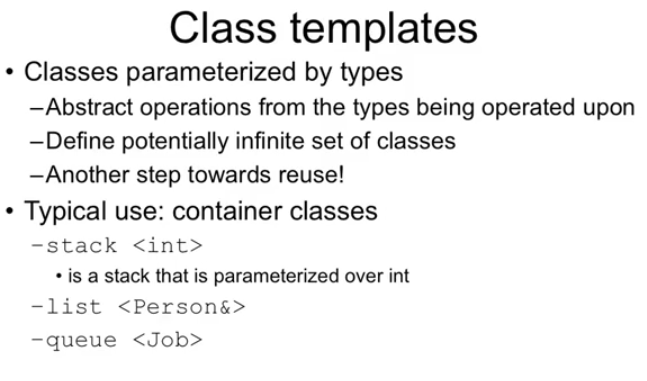
函数模板的初始化

1. 由编译器对传入的参数类型进行推定
2. 也可以显示地指定，如果参数没有出现在函数签名中。

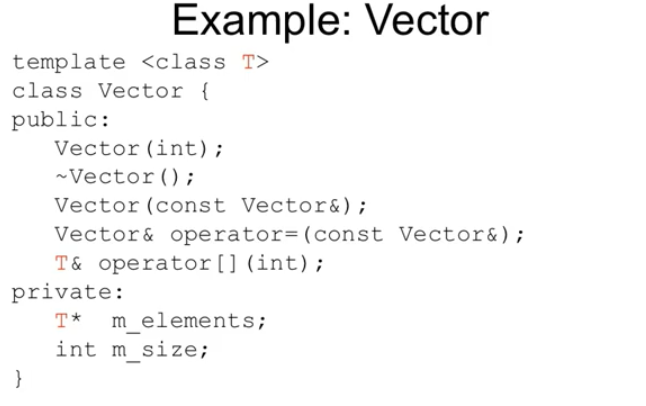
template < class T >

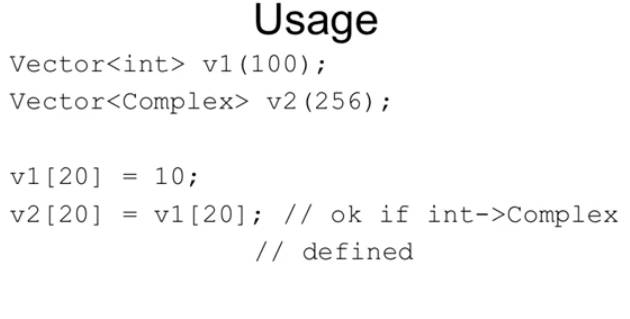
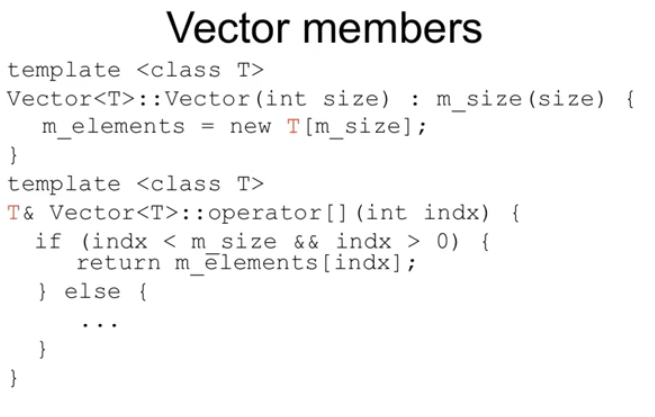
void foo( void ) {}

foo<int> (); // type T is int



**类模板**

2



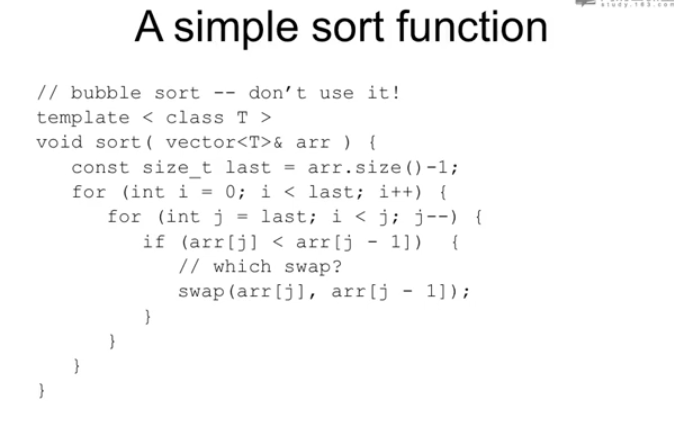
每个成员函数前面都有 template <class T> 这一部分，用以表明这是一个 **类模板** vector<T> 的成员函数，且定义T为类型参数（代表未知类型）。

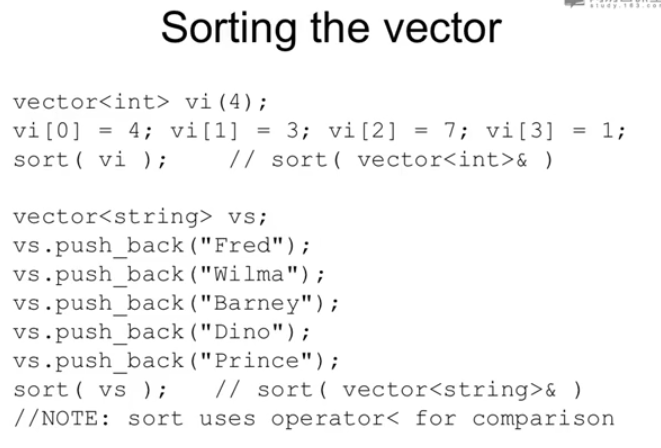
template < class T >

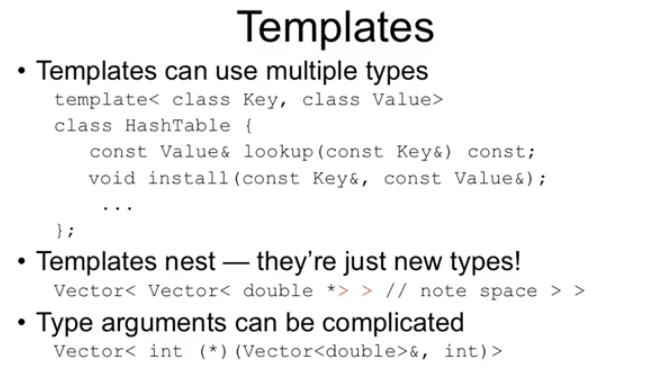
Vector<T>::Vector(int size) : m\_size(size){

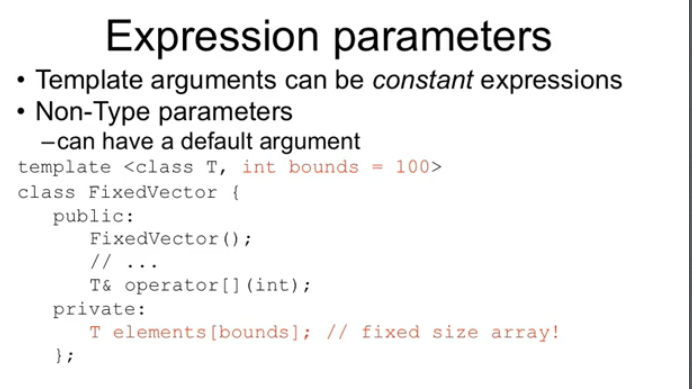
}

# # 35 模板Ⅱ



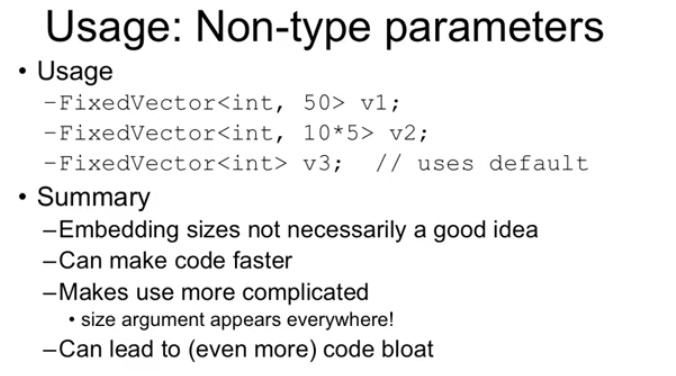


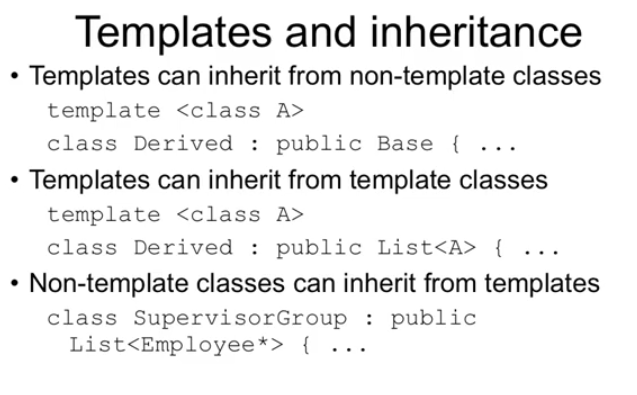




表达式参数

* 模板参数可以是 constant 表达式 **常量表达式**
* 非类型参数





* 模板可以从 **非模板类**继承
* 模板可以从 **模板类**中继承
* 非模板类 可以从**模板**中继承